HOCHSCHULE TRIER
Trier University of Applied Sciences
Informatik - Computer Science

Mobile Anwendung für die Kostenschätzung mit Android

Mobile Application for Cost Estimations in Android

Oliver Fries

Bachelor-Abschlussarbeit

Betreuer: Prof. Dr. Georg Rock

Trier, 29.02.2016

Abstract

Die Wichtigkeit von Kostenschätzungen von IT-Projekten steigt durch immer komplexere und größere Projekte stetig an. Schwierigkeiten bei der Kostenschätzung liegen im Zugriff auf bereits vorhandene Schätzungen, die zur Verbesserung der Schätzwerte beitragen, sowie der Auswahl relevanter Projekte, damit nicht alle verfügbaren betrachtet werden müssen. Diese Arbeit zielt darauf ab, den Prozess der Kostenschätzung als mobile Anwendung umzusetzen und die genannten Probleme mit einem dynamischen Prozess durchzuführen. Hierbei steht die Umsetzung des Function Point-Verfahrens im Fokus, sowie die Möglichkeit mit einem Vergleich von abgeschlossenen Projekten auf deren Schätzung zuzugreifen. Diese mobile Anwendung wurde als Android Applikation umgesetzt und trägt den Namen *MobileEstimate*. Es ergeben sich dadurch neue Möglichkeiten für Projektmanager und Projektteams zur schnellen und einfachen Kostenanalyse bei IT Projekten sowie zum Monitoring der Projektkosten.

The importance of cost estimation in IT projects is constantly increasing through more complex and larger projects. Some of the difficulties in cost estimations are the access to existing estimations, which help to improve the estimation, and the selection of relevant projects in order to not consider all available projects. This paper aims to implement the cost estimation as a mobile application with a dynamic process to approach the specified problems. Implementation of the Function Point method and the possibility to compare projects and get access to their cost estimation is the focus of this paper. The mobile application was implemented as an Android application and is called *MobileEstimate*. This results in new opportunities for project managers and project teams for fast and easy cost analysis for IT projects and the monitoring of project costs.

Contents

1

Introduction

Most of the contracts IT companies subscribe are projects and these are notorious for going past their deadline and over their budget. According to the study of Capgemini in 2014 [DU14], the importance of cost estimation increases every year. The study asked for the most important requirements in the IT for the next years. Top requirement, as asked in the study, is to increase the efficiency, which means to lower the costs and to meet determined deadlines. This will increase the effort companies have to take in planning their projects.
All businesses also want to lower the risk of delayed or canceled projects. This means IT companies have to take more effort in requirements engineering and cost estimation to give their clients an accurate estimation of the upcoming project. This results in a increasing effort for requirements engineering in IT projects. As cost estimation is a part of requirements engineering, it will lower the chance of a budget overrun. To achieve this 6% to 12% of the project time has to be spend in requirements engineering. The budget overrun is limited with this spent time to a maximum of 50% of the estimated cost[PA10]. Which means, that a higher effort in requirements engineering and cost estimation will lower the chances of failed projects, but to much effort has not more impact.
Therefore cost estimation is an important element for planning software projects and can be responsible for successful or failed projects. If a project was estimated right, budget overrun will be noticed early an retaliatory action can be executed. It is even more important to estimate as precisely as possible to guide the project to success. There are several methods for these estimations that can be used at different phases of the project. These methods of estimation lean their result on the information they get from the development process and the artifacts of the particular project phase. These include *requirement documents*, *diagrams* or the *program code* itself. All available artifacts depend on the used process model and the project phase [HU11]. Based on the described information, the actual project can be categorized, so that the *'best fitting'* estimation technique for the current estimation can be selected. These methods can be time-consuming and related projects can often only be found in the own company context or are based on experiences.
This paper aims to develop a mobile application which supports the Function Point estimation. The application aims to make the estimation process in IT projects

simpler and more efficient. To achieve a better way for estimating costs the most important design guideline was *'Only show what I need when I need it'* [GO16]. The comparison between projects has to be *formalized* and *implemented*. The idea is to give the user an overview over terminated projects and how they were estimated. The user has the possibility to transfer such an estimation to a new project or get a quick view how much man days it took.
The developed application *MobileEstimate* should prove that cost estimation on mobile devices is possible. It should allow to estimate the costs of a project with function point method and among the existing projects it should display *related projects*. Estimation results from a related project are possible to be viewed in the application and *transfered* to another estimation.

2

Theoretical Background

This chapter introduces the fundamental concepts such as the *cost estimation process*, a *state-of-the-art report*, different *estimation methods* and *techniques*. The cost estimation process in IT projects and the different methods to calculate the cost of a project are described in this chapter. The *state of the art* report combined with the *market description* will give a short overview over the situation about software estimation tools on the market and the possibility of a *mobile solution* for cost estimations. *Android* as the chosen platform and *Java* as the programming language will not be described in detail here and are assumed to be known.

2.1 Cost estimation in software engineering

The most expensive components of computer systems are software products. While private clients are mostly interested in the final price of a product, business clients of IT companies typically want to know the *costs* of the software **before** project launch. As analyzed in the *IT-Trends* study from *Capgemini* [DU14] the IT budget of companies is growing up to 10% every year. Whether developing a new project or standardizing existing software, minimizing the project costs is always a *main goal* of the project management. As human resources are the biggest part of software costs, project managers and especially business clients want to know the estimated spendings and completion time of a project, which derives the needed manpower. Most of the estimation methods focus on this aspect and give the result in *man days*. These estimated days can then be converted into the real costs by calculating the daily rate with the estimated time.
Basically the cost estimation in software engineering wants to answer following questions:

1. How much effort is required to complete the project?
2. How much days are needed to complete the project?
3. What is the total cost of the project?

While projects are a living thing, the effort may change due to unexpected difficulties. For a precise estimation of the total costs, an adjustment cannot be avoided and it can be useful to change the estimation method in a later project phase.

This means that the estimation process is not an *one-time* thing but will change through the life-time of a project [WI05].

2.1.1 Estimation Process

It is common to create the first cost estimation before the system design, but also for monitoring purposes, milestones or if the client wants an overview of the project. Each time an actual cost estimation is needed the estimation process is executed which is a set of techniques and procedures that are used to derive the software cost estimate. Kathleen Peters described the basic process of an estimation, as seen in figure 2.1, as it is common in the industry [KA16]. As can be seen in the figure, there are seven steps in this estimation process. The first part is to collect the *initial requirements* which is essential to know what the project is about and evaluate the approximate project size. With an selected estimation method, which are described in section 2.1.3, the evaluated size of the project is then estimated. Afterwards the effort in man days is calculated from which the cost schedule is created. Data from older projects can be included into the cost estimation which allows a more precise estimation. In the process step to *approve the estimation*, it has to be decided if the costs are acceptable or if the range of functions has to be shortened and the re-estimation has to be started from the beginning. If the cost estimation is acceptable the development of the product can start or continue.

In this classical view of the estimation process there are four outputs generated:

1. *Actual Size* - the size of the project as a numerical value to make it comparable.
2. *Manpower Loading* - the amount of personnel that is allocated to the project.
3. *Project Duration* - the time that is needed to complete the project.
4. *Effort* - the amount of effort required to complete the project is usually measured in units as man days (MD) or person months (PM).

As described before, the estimation process can be triggered at any time in the project to re-estimate the costs. Depending on the project stage another estimation method than used before can be more precise.
The overview shown in fig. 2.2 shows that the *SLIM* method is more suitable at the beginning of a project, whereas the *ZKP* method is more suitable after the system design stage[1]. Most of the estimation methods are useful after the study stage. This is because a rough overview of the project size exists afterwards. Different estimation methods may also change the *evaluation output*, which is one of the difficulties of cost estimations.

2.1.2 Difficulty of estimations

One of the problems in estimating costs is that the actual code is only a small part of the project. Beside project planning there are many administrative tasks

[1] `http://winfwiki.wi-fom.de/index.php/Methoden_und_Verfahren_der_Aufwandsch%C3%A4tzung_im_Vergleich`

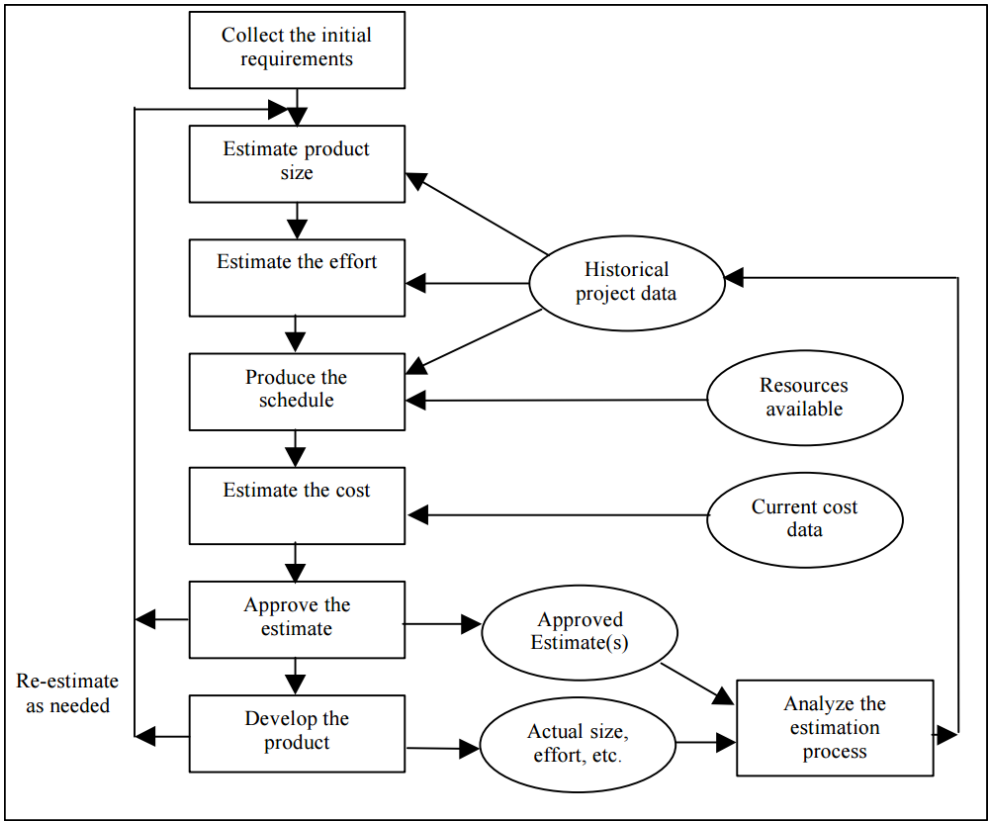

Fig. 2.1. - The Basic Project Estimation Process
Source: Peters, Kathleen - Software Project Estimation, Page 3

to do, like coordination of the project or searching and fixing errors. Most estimation methods evaluate the estimated time for the implementation only and too little or no time is evaluated for the non implementation tasks. This resolves in an underestimation of the non implementation tasks or an overestimation if they are predicted as a high value [WI05].
Most project managers rely on their experience from past projects. This is an advantage, but as technology changes fast and new projects inherit new problems there is no prior experience for some parts of the project which is another estimation difficulty. This can make experiences from prior projects useless. Because of the unique nature of projects it is common that a new project has big parts where no experience exists. Another difficulty of estimations is the fact that people who are inherited in the project have a more positive outlook and mostly underestimate the costs. Also, customers often got a target time for the project, which leads in adjustment of the cost estimation. This leads to budget overrun if the needed resources are not available for the project [WI16]. It is also not guaranteed that the estimated costs are accurate and stay within the budget. An estimation can easily go past their estimated target as new technologies and unexpected difficulties are commonplace. A partial requirements engineering can also cause an inaccurate

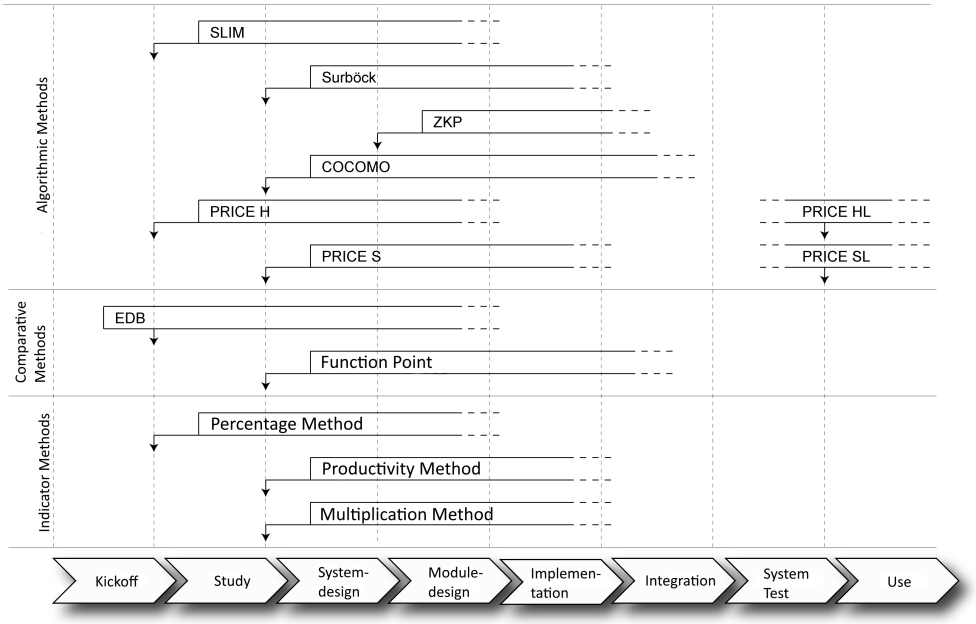

Fig. 2.2. - Starting Points of Estimation Methods

cost estimation due to unexpected difficulties in the implementation.

2.1.3 Methods for estimation

Estimation methods are different metrics to calculate the costs of a project and there are different approaches to categorize the estimation methods. The categorization used in this paper is based on the book *'Management von IT-Projekten'* by Hans W. Wieczorrek, where the methods are subdivided in *algorithmic*, *comparative*, *key figures* and *expert discussion* method[WI05].
All estimation techniques have in common, that only with a combined use of these different estimation methods a suitable result can be achieved as a measured value for the project. For the evaluation of the needed effort the underlying metric describes how this calculation is done. On the basis of charge rates the effort size is calculated out of it.

Algorithmic Method

The algorithmic method uses a closed formula, which is based on empiric evaluation of already terminated projects or on existing mathematically models. Different forms of this method are the weight and the sampling method, which only differ in their usage.
The accuracy of the estimation depends primarily on the precision of the influence factors [WI05]. The algorithmic method always connects measurable project sizes, such as *lines of code* and *implementable features*, with influencing factors to get

the result, represented as required effort in personnel costs. The basic formula for this method, as described by Wieczorrek [WI05]:

$$Personnelcosts = f(resultquantity, influencingfactors) \tag{2.1}$$

Comparative Method

Not based on a formula or numerical connection, the comparative method tries to create a reference between the current project and past projects. Therefore projects from the own company or the same industry sector are analyzed with appropriate comparison methods. This estimation method has the advantage, that it can be used early in project development [WI05]. This method can be used for hardware and software projects.

Key Figures Method

Estimation methods based on key figures can be differed to *multiplier* and *percentage* method. The *multiplier* method uses units of power as the base to estimate the total expenditure, whereas the *percentage* method uses the effort of a project stage to estimate the effort for the next stage.
After project completion, a post calculation determines the total project costs and the amount of specific types of costs. In order to calculate these costs, they will be divided by the scope of the developed product. This results in new key figures which can be used for new projects by multiplication of the estimated scope with the appropriate key figures. Regular actualization of the key figures is necessary for right results [WI05].

Expert Discussion Method

As a quantitative and heuristic method the expert discussion uses knowledge from selected groups of people. It differentiates between four kinds: *single person interview*, *multiple person interview*, *Delphi method* and *assessment meeting*.
The advantage of this estimation method is that they are useful for all project types. But they have the risk of a strongly affection by subjective opinions and the experience of the interviewees. As a result, expert discussions should never be used for complete projects but only for sub projects [WI05].

2.2 Estimation Techniques

The existing estimation techniques rely on experience-based judgments by project managers who created, by combining estimation methods, techniques that can be used to estimate the effort of a project. Most of the estimation techniques became popular in the 80's. With the agile projects becoming more popular, estimation

techniques for these project types are also rising.
Because of the fundamental uniqueness nature of projects an *'universal, everywhere applicable and always delivering the correct estimation'* technique does not exist, according to Litke [LI07]. There is also no clear selection progress for the estimation technique to use, beside the time aspect when the use of a technique is available. As shown in figure 2.2, the *Function Point* and the COCOMO technique are both possible after the study stage. As a comparative method, the function point technique has not much in common with COCOMO, which is an algorithmic technique. The project manager has to balance the weigh of each technique and probably choose that one he has most experience with. As an example for estimation techniques these will be described here in more detail with a comparison at the end.

2.2.1 Function Point

The Function Point technique was first mentioned by Allan J. Albrecht in 1979 at the IBM symposium [AL79]. He declared that an useful measurement of productivity is only possible in relation to the functionality that is visible to the user. This measured productivity needs to be independent of the used technology and is calculated with the proportion of project effort and the allocated function points. This resulted in the idea to turn over this calculation for a preliminary estimation of the effort the project would have. Because of the clarity and the flexibility the technique spread fast.
It helps to estimate the scope of a project to an early stage and is suitable for benchmarking in the own company as well as on national or international level [PO12]. It contains algorithmic and also comparative methods. Basically, this technique uses five steps for estimation [JE01]:

1. Determining the components
2. Evaluation of the components
3. Calculating the function points
4. Categorization of the influence factors
5. Calculating the development effort with the function points and influence factors

The most important part of this technique is that all measurements only include the user view. This means that the user view focuses on the functions that are important for the specific business process. Implemented business processes are the components that have to be determined.

Determining the Components

To divide the project in useful components, all inherited business processes are divided into elementary process.This distinction is made into five categories:

1. Input Data
2. Output Data

3. Request
4. Dataset
5. Reference Data

The *Input Data*, sometimes called *External Inputs* (EI), is an elementary process in which data crosses the boundary from the outside of the application to the inside. This data comes from a data input screen or another application. Maintaining one or more logical files it is used to control business informations.
External Outputs (EO), or *Output Data*, are an elementary process in which derived data passes across the boundary from the inside to the outside. Created output files or reports are sent to other applications. These are created from one or more internal logical files and external interface files.
Internal *Logical Files* (ILF's) or *Dataset* are an user identifiable group of logically related data that resides entirely within the applications boundary and is maintained through external inputs.
The next category are *Requests* or *External Inquiry* (EQ). An elementary process with both input and output components that result in data retrieval from one or more internal logical files and external interface files. This process does not update any *Internal Logical Files*, and the output side does not contain derived data.
The last category is the *Reference Data* or *External Interface Files* (EIF), which are a user identifiable group of logically related data that is used for reference purposes only. This data resides entirely outside of the application and is maintained by another application [LO16].
These are the smallest and from a business perspective useful and closed activities, that can be performed by the system [PO12]. It is useful to categorize the components, because a change in the datasets is followed by more effort than changing a request [WI05]. After the categorization there is an amount of items for each category which has then to be evaluated in the next step.

Evaluation of the Components

The evaluation of the components is simply the classification of each category in their level of difficulty (simple, medium or complex). After this, each component is multiplied with the point value according to their difficulty. The table 2.2.1.1 is described by Wieczorrek and are standard values for the function point technique [LO16].

Calculating the amount of Function-Points

After the evaluation there is an amount of components for each category. To get the number of function points each category has to be multiplied according to the selected weigh and all categories are summed. This results in the equation 2.2.*Function* is the respective category and *Difficulty* is the weigh from table 2.2.1.1. The resulted value *E1* is necessary for evaluating the estimated points with the influence factor.

$$E1 = \sum_{1}^{n} (Function \cdot Difficulty) \tag{2.2}$$

Category	simple	medium	complex
Input Data	3	4	6
Output Data	4	5	7
Request	3	4	6
Dataset	7	10	15
Reference Data	5	7	10

Table 2.2.1.1. Point value of each category

Classification of Influence Factors

Do get a more realistic estimation, all influences that can affect the project surroundings are measured. There are seven defined influence factors [BA08], which are described below. Each of the influence factor has a value between zero and five which describes how much the factor influences the project. Exception are *Arithmetic Operation* and *Exception Regulation* which go from *zero* to ten.

1. **Integration into other applications**
   This factor describes if the system will work with distributed data. Zero means that the system does not work with other applications and five means that there is an integration into other applications in both ways.
2. **Local Data Processing**
   The system will work with different applications and will send and receive data from other applications. This states if there is a cooperation with other applications and if the communication exists online or offline.
3. **Transaction Rate**
   A high transaction rate affects planing, development, installation and maintenance of the system. It describes how much transactions are expected with the system.
4. **Processing Logic**
   The processing logic is divided in 4 subcategories: *Arithmetic Operation*, *Control Procedure*, *Exception Regulation* and *Logic*. *Arithmetic Operation* describes the intensity of the operations in the project. The controlling of the results is stated within the *Control Procedure*. The *Exception Regulation* describes how eventual exceptions are treated and the *Logic* multiplier describes how much effort is to be expected for planning the logical component of the project.
5. **Reusability**
   How much of the produced software has to be reusable in other projects is described with this category. A high value means extra effort in the planning stage for module-based development.

6. **Stock Conversion**
   This describes how much of the used data needs to be transformed for the usage within the project. A high value means, that much input data from other applications have to be transformed into data the application can process.
7. **Facilitate Change**
   The application was especially planned and developed in such a way that changes can be made easily. A high value means that the user can make changes on the system on its own and that the changes are available immediately.

When all influence factors are set, all values for each factor will be summed up to the value *E2*, as described in the following equation.

$$E2 = \sum_{1}^{n} \textit{Influence Factor} \tag{2.3}$$

This influence factor indicator has then to be transformed to a multiplier that calculable with function points [BA08][LO16].

$$E3 = \frac{E2}{100} + 0,7 \tag{2.4}$$

Calculation of the Function Points

With the calculated Function Points (*E1*) and the Influence Factor multiplier (*E3*) the *total-function-points* (TFP) can now be calculated with the following formula:

$$TFP = E1 \cdot E3 \tag{2.5}$$

These *TFP* have no unit and represent only points. They have to be calculated into man days with a *points-per-day* value. From the regression analysis of previous projects a standard calculation of Function Points per day can be made using table 2.2.1.2.
The project has to be classified with their Total-Function-Points and divided through the appropriate points per day, as described in 2.2.1.2. This results in the expected man days for this project.

Estimated Size	Function Points	Points per Day
Small Project	≤350	18
Mid Small Project	≤650	16
Medium Project	≤1100	14
Mid Large Project	≤2000	12
Large Project	≥2000	10

Table 2.2.1.2. Function Points to Days

2.2.2 COCOMO

The Constructive Cost Model (COCOMO) is mostly used when it's not possible to rely on experience. It is based on algorithmic and parametric methods that merge parameters from *software projects*, combining the *system size*, *product properties*, *project* and *team factors* with the *effort for developing* the system [JE01].
As an public domain model it is free to use and is considered to be a classic for algorithmic methods. This technique was adjusted to the IT development through the years and is common in many companies. The accuracy of the COCOMO techniques rises with later project stages. The deviation of the cost estimation is at the beginning of the project between $0{,}25 \cdot MD$ and $4 \cdot MD$. In later project stages this variation decrease until it is zero just before the project ending [SO07].
The parameters for COCOMO can be split into three parts. The *project classes*, *model variants* and the *implementation time and effort*.

Project Classes

The *project classes* represent the estimated size of the program itself. These are expressed in *kilo delivered source instructions* (KDSI). COCOMO classifies three project classes with different calculation factors as described in table 2.2.2.1 [SO07].

Complexity	Calculation Factor	KDSI	Describtion
Small	1.05	< 50	A small, well-known project team works together, the environment is well-known, there is no big innovation necessary and no pressure due to a deadline.
Medium	1.12	50 - 300	Employees with average experience, team members with some experiences in different parts of the project are working together.
Complex	1.20	> 300	High cost and deadline pressure, high innovations and an extensive project. High requirements to the project team and new components.

Table 2.2.2.1. COCOMO project classes

Model Variants

The COCOMO technique can be differed into *basis*, *intermediate* and *detailed model*. These represent the detail level of the cost estimation.
The first stage is also called *basis estimation* and is a rough estimation, which estimates the project costs to an early stage of the project. The costs are calculated

with an equation without dividing the project into structure or time aspects. The base model is an useful starting point for later estimations.
The second stage is the *intermediate* model and adjusts the first estimation to a higher level of detail by differentiating the development stages. It is not a parametric estimation yet, because not all data can be considered.
The *detailed model* is the last stage of the estimation and allocate the estimation with fifteen influence factors [JE01]. These influence factors are divided into four categories.

1. **Product Attributes**
   a) Required Software Reliability
   b) Size of the Application Database
   c) Complexity of the Product
2. **Hardware Attributes**
   a) Run-time Performance Constraints
   b) Memory Constraints
   c) Volatility of the Virtual Machine Environment
   d) Required Turnabout Times
3. **Personal Attributes**
   a) Influences Analyst Capability
   b) Software Engineering Capability
   c) Applications Experience
   d) Virtual Machine Experience
   e) Programming Language Experience
4. **Project Attributes**
   a) Use of Software Tool
   b) Application of Software Engineering Methods
   c) Required Development Schedule

Calculation of Implementation Time and Effort

Because of the differences in the models of COCOMO estimations, each model has its own equation for cost calculation. Each formula calculates the estimated time of the project in person month (PM), which are stated by Boehm as 152 working hours resulting in one PM, with 19 working days per month and eight hour days [BO81].
The formula for the basic model calculates the complexity of a project with the KDSI value and the calculation factor of the project which gives the calculated PM as a result. The formula is:

$$PM = Complexity \cdot (KDSI)^{Calculation\ Factor} \tag{2.6}$$

The *KDSI* value is the expected amount of code lines in the project. As described in table 2.2.2.1, the Calculation Factor is derived from the *KDSI* value. To figure out the complexity multiplicand table 2.2.2.2 describes for each project the suitable

solution.
The equation for the *intermediate model* is the same as for the *basic model* (2.6) [BO81]. The difference is the changed multiplicand for this model, as described in table 2.2.2.2. The detailed model is an extension of the intermediate model that adds effort multipliers for each phase of the project to determine the cost drivers impact on each step. The effort is calculated as function of program size and a set of cost drivers given according to each phase of software life cycle [BO81]. For an equation it is necessary to multiply all cost drivers C_i together, as described in the following formula:

$$C = \prod_{i=1}^{15} C_i \tag{2.7}$$

The product of all 15 influence factors C_i is the result of this equation, which gives a combined influence factor which can be multiplied with *KDSI* value to get the *total time for development* (TDEV). But therefore the *KDSI* has to be calculated with the complexity factor for the detailed model from table 2.2.2.2. This calculation can be described in the following equation [BO81]:

$$TDEV = C \cdot (KDSI)^{CalculationFactor} \tag{2.8}$$

Complexity	Basic Model	Intermediate Model	Detailed Model
Small Project	2.4	3.2	0.38
Medium Project	3.0	3.0	0.35
Complex Project	3.6	2.8	0.43

Table 2.2.2.2. COCOMO complexity multiplicands

2.2.3 Comparison

The Function Point analysis is useful for software projects of all size, but mainly desktop based platforms. This is one of the biggest contrapositions as this technique is not useful for console programs. Instead COCOMO is used for large corporate and government projects, including embedded firmware projects. The major source of criticism at COCOMO estimations is that they are inappropriate for small projects and that it is based on the waterfall model. It is recommended for large projects and project with none experience of the team to use COCOMO 2 instead.
The similarities between these models are that they both rely on algorithmic methods, with the difference that COCOMO is a logarithmic type and function point is linear. Both were created in the same time, where the project development cycle relied on linear procedure models and the used programming languages where

procedural. None of the techniques is necessarily better or worse than the other, in fact, their strengths and weaknesses are often complimentary to each other. There is no estimation technique that is the *'best fitting'* for a project [AB14].

2.3 Software for Cost Estimations - State of the art

The big uncertainty of cost estimations is an actual problem. Stake holders and project managers don't trust the estimation output and calculate a surcharge on all estimations. In big projects this uncertainty factor, from 5% up to 50%, is added on the estimated costs for a project [FI10]. That is because many large projects are stopped due to budget overrun or incomplete requirements [ST14], which results in insecurities on the cost estimation output.
Whereas the cost estimation is possible as paper work, there are several software products that allow it. As many start-up IT companies have been founded in the last five years, none of them develops cost estimation software [RI15]. There is also no approach in developing a mobile application for cost estimations. The use of smartphones increased from 2013 to 2014 by 21% and 80% of the people in the business environment use a smartphone [LO14]. This opens up good opportunities on the market for the application.
There are three cost estimation tools that are mostly referred to which are introduced in this section.

2.3.1 SEER - Cost Estimation Software

The *'Seer - Cost Estimation Software'* is developed by Galorath Inc. in Los Angeles, USA. Galorath Inc. offers consulting and software for *cost estimation*, *decision support* and *project management*. The company was founded in 1979 with the goal to improve the software and hardware development process in the industry. The next step was to improve their consulting quality by developing their own tools for this process. *Seer* is the name of their tool set with a large variety of different tools that support the development process of new products.
Seer for Software is an estimation application for *'estimating, planning, analyzing and managing complex software projects'*[2]. Based on the *SEER design principles* the software contains an annotated and guided interface for defining projects, a parametric simulation engine and numerous standard and custom reporting options. Due to an open architecture *API*, the *SEER* application can be integrated with enterprise applications and departmental productivity solutions. *Galorath* specifies that all estimations within this software are repeatable and consistent.
According to the software description, *'a high-level software estimate can be developed in a matter of minutes using SEER´s intuitive, window-based interface'* [PR15]. A dialog guides the user through the process of creating a new estimation.

[2] `galorath.com/products/software/SEER-Software-Cost-Estimation`

In the first screen the user has to set a project name and decides whether he creates an empty project or starts with a scenario. The scenarios are example projects with some data for starting the estimation. In the next step the user chooses the estimation technique. These are subdivided in *functional*, *lines* and *sizing scale*. Another feature of the software is the documentation and export function. All estimations can be exported to *Microsoft Project*, *Microsoft Office*, *IBM Rational* or other third-party software. SEER delivers a huge variety of estimation techniques, guided processes and many possibilities for documentation and reporting of all estimations.
The software can be bought in an *estimator*, *project manager* and *studio version*, whereas the *estimator version* inherits only a standard estimation. The *project manager* and *studio version* allow estimation checking and access to the projects database. The *studio version* allows also independent crosscheck and verification. The price of each software version is not specified on the homepage and must be requested.

2.3.2 PRICE - Cost Estimation Software

PRICE Systems L.L.C. provides agile estimating solutions. With their head office in Maunt Laurel, USA, and 12 locations worldwide they offer estimating acquisitions[3]. Their *Software Development Cost Model* is one of the oldest and widely used software parametric models for software development projects. In 2003 *PRICE* released *TruePlanning*, which contains methods that estimate the *scope*, *cost*, *effort* and *schedule* for software projects.
For cost estimation, purchasing efficiency and budget planning *PRICE Systems* develops *'multi-faceted cost estimating solutions'* [PR15]. Their biggest project is the *PRICE Estimating Systems* (ESI) framework that delivers solutions for estimating projects and cost management. This framework is not specialized for estimating only software projects but also other projects.
The homepage describes that it is possible to estimate projects with all current *state of the art* cost estimations. A collaboration with other users is also possible as well as sharing the results through the program. This allows faster teamwork and a better estimation output. A data driven method for all estimations allows to compare the estimation with already completed projects. *TruePlanning* allows also estimation output to a specific work breakdown structure or cost element structure for accurate *top-down* and *bottom-up* estimate comparisons. Mappings can be stored, retrieved, and modified to keep pace even as program activities change. Beyond specific cost estimating products and services, *PRICE Systems* offers strategic cost management services. They collaborate with estimators, engineers, project managers, and financial and executive management to design and implement integrated cost management systems that meet the unique challenges in the environment.
Any details about the cost of the software or licensing have to be requested at *Price Systems*.

[3] `http://www.pricesystems.com/`

2.3.3 SLIM Estimate

SLIM Estimate is a cost estimating software developed by *Quantity Software Management*[4] (QSM), an estimation company with their head office in McLean, USA. *QSM* was founded in 1978 as a software management consultant company for measurement, estimation and controlling projects. Beside their consulting business segment they develop the *SLIM* Software which contains software for controlling, metrics and estimation of projects.
The *SLIM Estimate* software provides a flexible cost estimation which aligns the estimation to the project size. Integrated schedules offer simple exports of an existing calculation. The software can suggest alternate solutions calculated on past projects.
The homepage does not go into detail how this suggestion works. For a new estimation the user can use the *SLIM Database* and the *QSMA* database, which can be used for new estimations.
For better use in the company the *SLIM Estimate* inherits interfaces to *MS Office* and the possibility export estimations as a web representation. *SLIM Estimate* delivers hereby a software package with many functionalities. Especially the access to a large database and the knowledge base of *QSM* is a big benefit for the application. As a result, it is possible to access the experience of past projects and get the benefit from this experience.

2.3.4 Commonalities

All described cost estimation tools have in common, that they inherit different estimation techniques. The user can always decide which technique he wants to use. They are all tools that are on the market for many years and rely on experience from project managers. Each software has a different approach on the implemented estimation process and collaboration of project estimations. They all have a database with previous cost estimations but with extra costs for accessing it.
The User interface is like in many large software projects, more functional and does not contain a modern design. None of them has an approach for developing a mobile application and rely on Microsoft Windows as their operating system.

2.3.5 Kinvey

A special software developer is *Kinvey* from Boston whose main business is the development of mobile applications. They don't develop a certain cost estimation tool, but in terms of general cost estimations, the company is still very interesting. *Kinvey* offers an *online estimation* for mobile applications in which the potential customers can estimate the costs for their application[5]. The estimations show the time the customer would have to spend for development and the costs for

[4] `http://qsma.com/slim-estimate`
[5] `http://www.kinvey.com/mbaas-savings-calculator`

development at *Kinvey*. The cost of developing through *Kinvey* here are always cheaper than a proprietary development.
The cost estimation process is simple. The estimation schedule is a one-page where the user can select the components of his application. The user is guided through some question about features the application should contain. After each step the user can see the actual cost estimation of the project. The costs are calculated from the selected components in man days. It is not possible to define and add your own components for the estimation. The individual items that the user can choose are sorted by groups and displayed graphically. Estimating the cost for a mobile application development can also be used for own estimations, but there is no way to export this estimation.

3

Application concept

This chapter describes the project plan and the concepts for the developed application. These concepts are the foundation of the developed application and describe how the internal processes in the application works.

3.1 Project planning

The project planned and performed with the *incremental development model*. For initial planning the features from existing cost estimation software were evaluated, as described in section 2.3. The next step was creating a *GUI prototype* to show how the estimation would look like on a smartphone. Afterwards each cycle followed the rules of incremental development. An evaluation of the wanted features for the application was done in meetings with my supervisor Prof. Dr. Georg Rock. Those requested features were compiled into the project goals and requirements. For the next step they were analyzed and implemented. Testing for the application were made afterwards and the result was then presented in a meeting. Further features were planned for the next iteration cycle began.

3.1.1 Project Goals

Project goals describe the benefit of the project and describe the condition to be achieved with the application. These goals were defined before starting with the definition of the requirements which are described in table 3.1.1.1. Most important goal is *G01* which describes that the cost estimation has to be implemented for *mobile device* use. As the developed application is only a beta version, it has to be developed modular for future feature implementation, which is described in *G04*. *G06* describes another important goal that is to be achieved which is finding related projects and compare them.

ID	Target
G01	The application must allow a quick and mobile cost estimation of IT projects.
G02	The application must provide information how cost estimations work.
G03	The application has to improve the cost estimation during the project life cycle.
G04	The application architecture has to be modular to add new cost estimation methods and analysis tools.
G05	With the application, it is possible to get the information of completed projects and take advantage of their estimation.
G06	The application allows comparison between projects and shows projects that are related to each other.
G07	The application gives information about estimated man days and the estimated costs of a project.
G08	A project analysis within the application allows an overview of the project changes and gives detailed information about the estimation.
G09	The function point method has to be implemented and is fully operational until the end of the bachelor thesis.

Table 3.1.1.1. Project Targets

3.1.2 Requirements

All requirements for the implemented application are based on the goals described in section 3.1.1 and from regular meetings. These are described in the appendix A and meet the demands for requirements as described by Balzert [BA09].

3.1.3 Cost Estimation

For the evaluation of the needed time, the project was estimated with the *Function Point* technique. Therefore all functionalities are grouped into **functional groups**, as described by Poensgen [PO12]. Afterwards each function was assigned a category for the estimation, as described in 2.2.1.
The group *Project*, as described in 3.1, inherits all functions a project has. All function groups for the application can be found in the appendix B. The project functionalities in 3.1 are created with the requirements. Some requirements can be grouped into one functionality. As an example the function *Create/Edit Project* inherits all requirements described in the appendix A for creation and editing of projects. This function is marked as an *EI* and *ILF* because all changes are made by the user for this function and creates or edit all internal files for the project. Functions that expects an input from the user are marked as *EI*, like `'delete project'`. *Output* functions like `'show estimated cost'` are a *EO* function, because they only display values and results to the user. Functions with a calculation are described as *EQ* combined with *EO* if they produce an output like

Export Project or process an *Input* like *Change Estimation*. This categorization of the function groups was inserted into the estimation table as described in table 3.1.3.1. Summed up the project has total **246** *Function Points*.

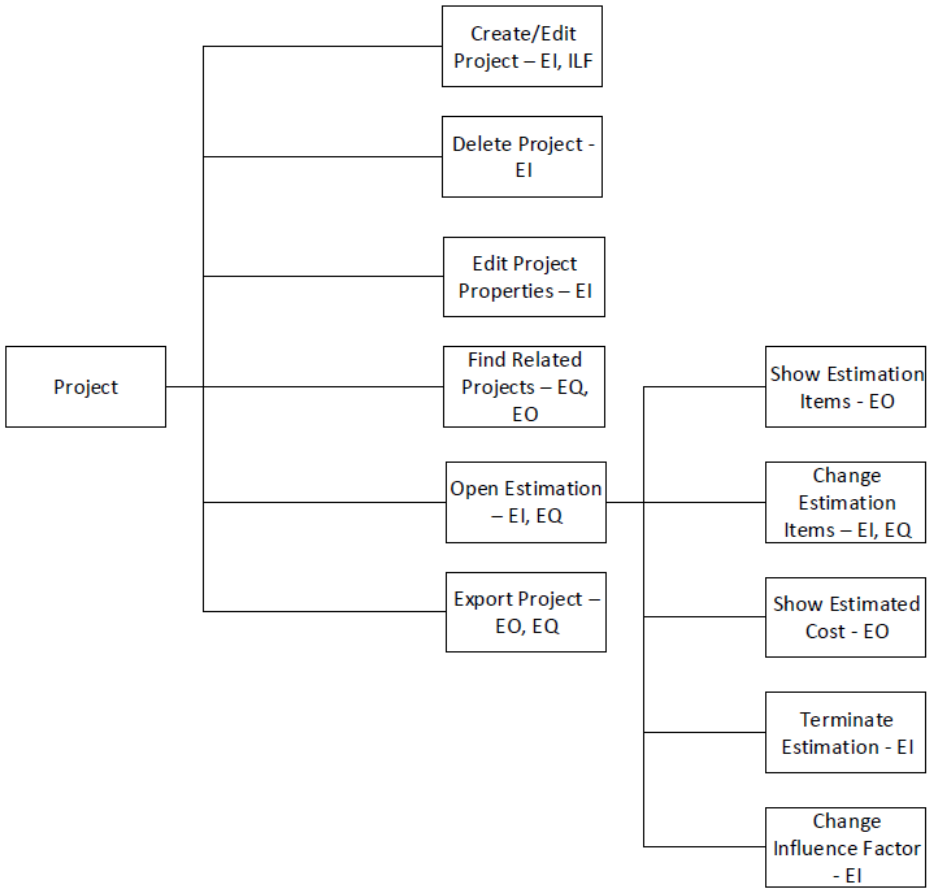

Fig. 3.1. - Function Group - Project

Category	Amount	Classification	Weight	Result of Row
Input Data	17	medium	4	68
Request Data	6	simple	3	18
Output Data	8	medium	5	40
Dataset	8	complex	15	120
Reference Data	0	simple	5	0
Sum				**246**

Table 3.1.3.1. Estimation - Total Points

As described in section 2.2.1, the next step is to set the influence factors which are described in table 3.1.3.2. The influence factor *Integration into other applications*

is set to two because the application has no direct communication to other software. The application does not work with other applications but has local data, like the databases that have to be processed, which is the reason for setting the factor *Local Data Processing* to two. As the application is only a local software for the end users device there is not much transaction to be excepted. To assure that there is enough time planned to implement an access to the database the factor *Transaction Rate* is set to one. There are many algorithms and equations in the application which have to be checked for errors and exceptions. Also a high effort for the logical component is to be expected. Because of this all influence factors in the *Processing Logic* group are set to the second highest value. For further development of the application a high effort has to be expected for *Reusability* in the project which is the reason for this influence factor to be *two*. There is not much data to be expected from other applications. Only the database causes a higher effort with queries to transform the data from the database in readable information for the application. Changes in the application can not be made by the user. He can only change settings and some appearances which is why the influence factor *Facilitate Change* is set to *one*. Calculated with the equation from 2.2.1, the influence factors result in a multiplier of **1.01**. Together with the *total function points*, the influence multiplier is calculated with the equation 2.5 which has **243.54** points as a result. Calculated with the points per day from table 2.2.1.2 the project is estimated to take **14** man days. In section 6 it is analyzed how this estimation has worked out and how much time the project took in reality.

Influence Factor	Weight
Integration into other applications	2
Local Data Processing	2
Transaction Rate	1
Processing Logic	
- Arithmetic Operation	4
- Control Procedure	4
- Exception Regulation	8
- Logic	4
Reusability	3
Stock Conversion	2
Facilitate Change	1
Sum	**31**

Table 3.1.3.2. Estimation - Influence Factors

3.2 Architecture

The architecture of the project is described as a component diagram in fig. 3.2. As usual in Android developments all resources are stored in the `resources` folder and are accessible from all other classes. The resources inherits `strings`, `images` and many more. Most used in the same folder is the *Layout Data*, which inherits all information for displaying the user interface. Layout data is connected with the *Activities* and *Fragments* component which are responsible to display the user interface.
The *Project Analyzer* component is responsible for collecting the data from all projects and putting them together for analysis. As the name says, *Project Data* contains all informations for a single project. For reading and storing data to the database, the component *Database Helper* allows simple *Data Requests* and *Data Insertions* to the database. It also allows a connection to the *XML Helper* which reads data that is stored to XML files. The *Project Exporter* component is called from an activity. It loads and compiles the data of a project for exporting it to files which can be used by external applications. Last component is the *Project Relation Solver*. It gets an project from an activity and searches related projects in the database.

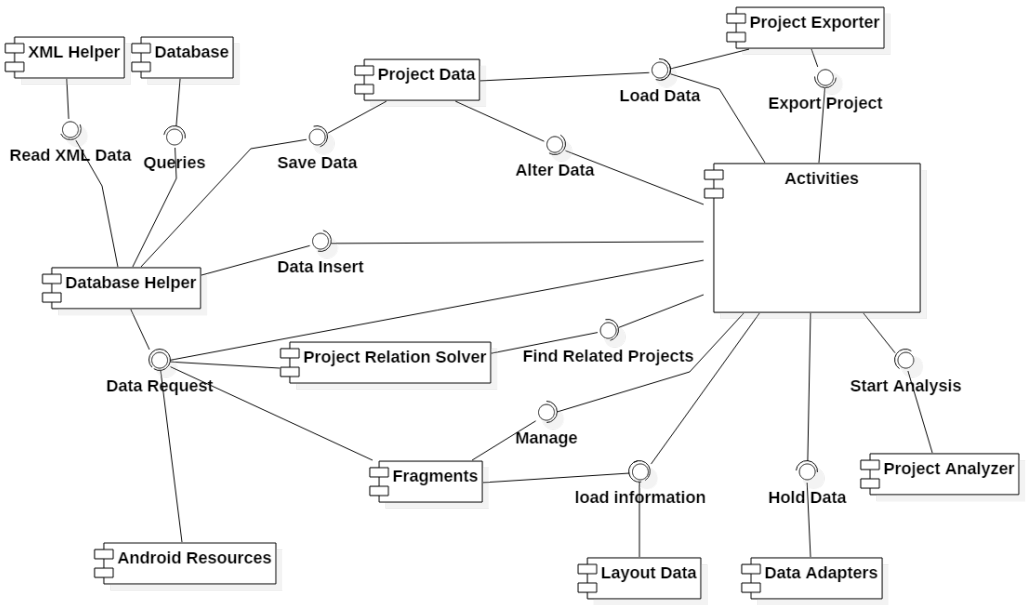

Fig. 3.2. - Component Diagram

3.3 Components

This section explains the components from figure 3.2 in detail and what concepts they inherit.

3.3.1 Database Helper

The *Database Helper* component is responsible for accessing the database, where all project informations are stored. Every *class* that needs access to the database should only create an *object* of this component and can `read` or `write` data to the database.
The component contains `name` and `path` to the database and allows the initialization. As described in the *sequence diagram* 3.3, with the message *Open Constructor*, an initialization of the database should be done in the constructor of classes that need access to it. This will open the *SQLite* database and ensures that requests on the database can be performed and no error occurs while accessing a non initialized database. All functionalities that are working on the database are in fig. 3.3 described, combined to the message *Using Class* and combines the three categories `SELECT`, `ALTER` and `DELETE` for working with the database.
For requesting data from the database a Java class has to send a request with the table name and the selection criteria to the Database Helper. This will be transformed into a query which is send to the database. Before sending this data back to the calling Java class the *Database Helper* has to transform the *SQL query* result into readable information.
Editing existing informations in the database is done with the message *Alter Existing Data* and the parameters table name and data to be edited. Before sending data to the database the *Database Helper* has to prepare it to insert them in the right column of the table. This query will return `true` if inserting was successful and `false` if it was not.
Last message option is *Delete Data* which will delete entries from the database. The parameter *tablename* and *Data ID* are the identifiers for a selected entry. Before deleting data from the database the *Database Helper* has to check if there are any other tables depending to the selected entry and has to ensure that they are deleted too if necessary. After all depending tables are identified the *Database Helper* will send the `delete` *query* to the database. It will then return a boolean value for the deletion process which is forwarded to the Java Class.

3.3.2 Project Data

The *Project Data* component was created to combine all data of a project from the database and make it accessible for other classes. It contains *Name*, *Description*, *Creation Date* and *Estimation Technique*. This makes it possible to get a fast overview of the different projects in the application. For estimation specific purposes the *Estimation Items* and *Influence Factor* are minor components for the *Project Data* and differ from the projects estimation technique. The *Properties* are also a minor component, which was done to build a module-based structure. As the output information of estimation techniques is the same, the main component contains the amount of *Estimated Man Days* for the project. If the project is completed the *Final Man Days* represent the time the project really took. This value is important to improve future estimations.

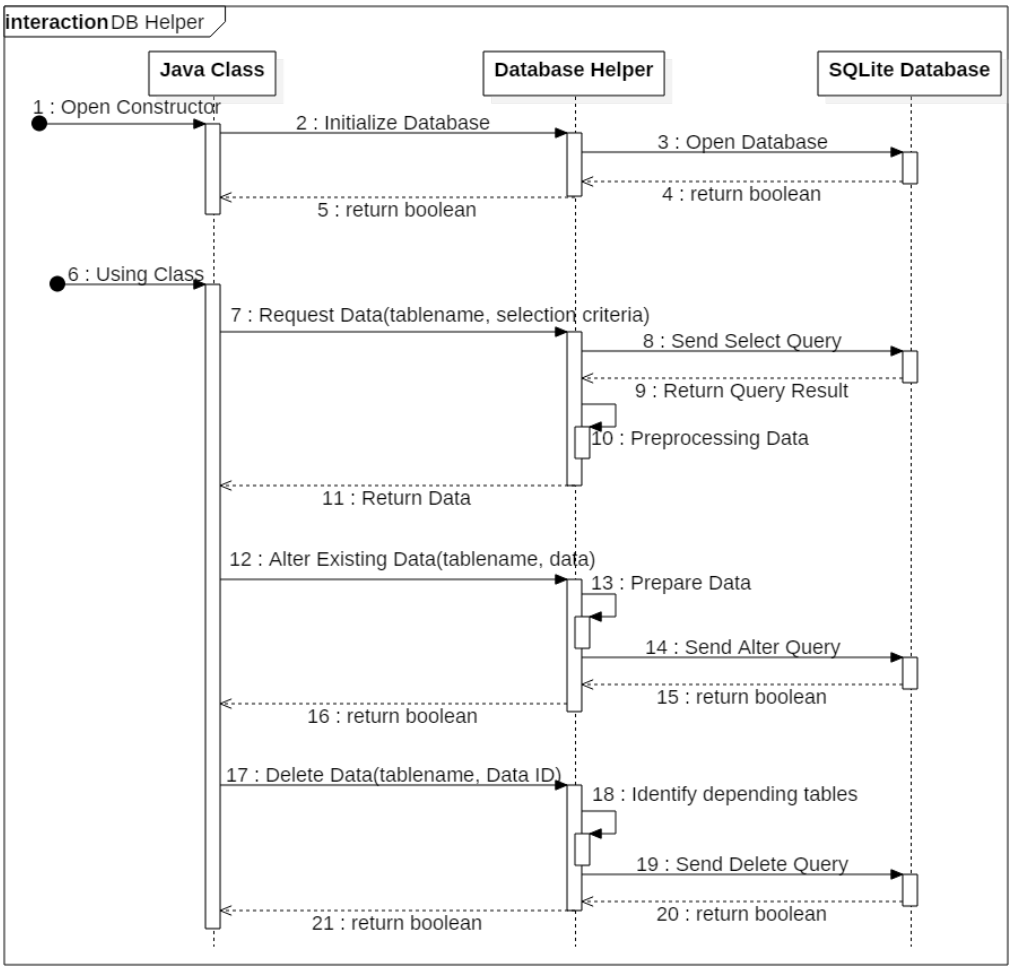

Fig. 3.3. - DB Helper Sequence Model

Project Properties

This minor component contains the information that is needed for the *Project Relation Solver* in section 3.3.4. It is a data component which stores all properties that allows reading and editing. Access to the properties is granted only through the *Project* component from section 3.3.2, which assures that a property is bound to a project.
The available properties are based upon the parts a project can be split to. Not every available factor of a project makes sense for categorizing it. For example a categorization into the *size* of the project makes no sense as this cannot be compared to a project that is not estimated or developed. Also a categorization of the costs makes no sense as it cannot be compared to a project that has to be estimated. For this reason the available properties are *Development Market*, *Development Type*, *Procedure Model*, *Platform*, *Industry Sector*, *Programming Language* and *Software Architecture*. These values categorize a project without saying something about the size or cost and makes it possible to compare with other projects.

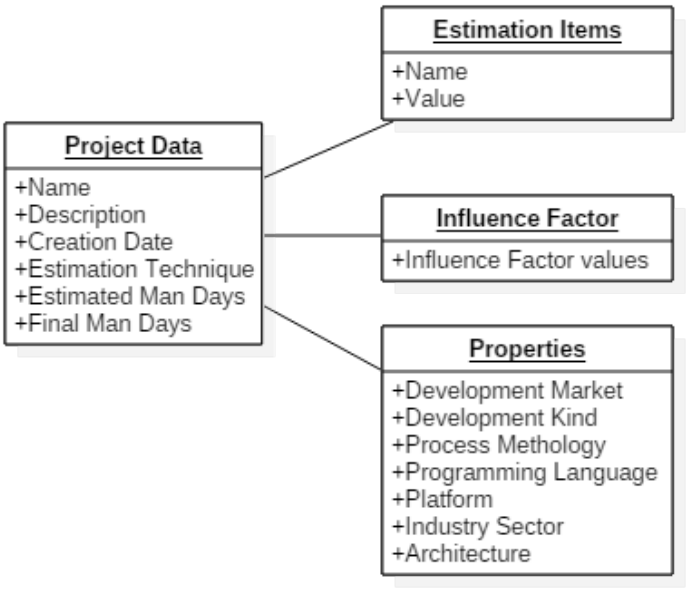

Fig. 3.4. - Object Diagram for Projects

All project properties contain only a selection of the possible values as a showcase. For each category further options are possible. Properties that have more options that are not implemented at the moment are combined in the option *other*. But this option is calculated with the highest possible distance at the moment. It should only illustrate that there are more options possible and allow a selection if the needed property is not available.

Estimation Items

All items for the estimation are in the *Estimation Items* components. These are the items for an estimation technique that are responsible for the estimation itself. In the *Function Point* estimation for example, these items calculate the value *E1* from the equation 2.2.

Influence Factors

The *Influence Factors* component contains all informations for *Influence Factors*. These are *name*, *chosen value* and the *limit* for the value. This component is also responsible for preprocessing the influence factor for usable information at the calculation of the estimation. In the Function Point estimation for example, this component calculates the value *E2* and *E3* that are described in section 2.2.1.

3.3.3 Function Point Estimation

The *Function Point* component is responsible for estimating a project. It inherits the calculation from section 2.2.1 and calculates the estimated man days. To get

the best calculation the component searches all existing projects in the database that are already terminated and are using the *Function Point* technique. In all of these projects the component searches for the project with highest total function points that are less then the selected project and for the lowest total function points that are higher than the selected project. This is visualized in figure 3.5. Next step is to calculate the *points-per-day* from these projects with the equation 3.1. If there is no project available for one or both of these value the component will instead take the base *points-per-day* value from table 2.2.1.2.

$$Points\ per\ Day = \frac{Total\ Function\ Points}{Final\ Man\ Days} \tag{3.1}$$

With these two values the component will then calculate the *average points-per-day* which is then used to calculate the man days of the actual project.

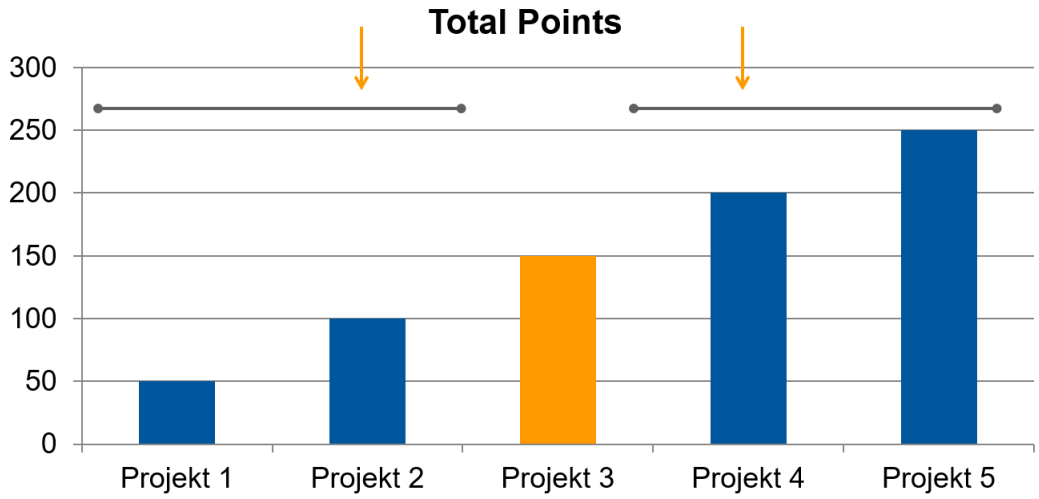

Fig. 3.5. - Visualization Points-Per-Day

3.3.4 Project Relation Solver

To fulfill the goal *G06* from section 3.1.1, this component is responsible to find related project. The point of this component is to compare already existing projects with a selected one. This should help the user to get an overview of how similar projects were estimated. Furthermore the *Project Relation Solver* should allow to transfer the estimation from a related project to the selected one to have an estimation where the user can build on. This comparison is made in mainly three steps *determination of the property distances*, *determination of the total distance* and *calculation of the relation*.

Determination of the Property Distances

Each project has *eight* different properties which are crucial for finding a relation. These properties are *Development Market*, *Development Kind*, *Procedure Model*,

Platform, *Industry Sector*, *Programming Language*, *Software Architecture* and the *Estimation Technique*. The first step for calculating the project relation is to compare each property and get the *distance*. Each of them has its own calculation for the distance and a different weight to fit in the *total distance* equation.

Estimation Technique

The selected *Estimation Technique* will influence the relation between projects. As it should be possible to transfer the estimation from a related project, this feature is not possible if these projects do not use the same estimation technique. So it is an important factor for the calculation. If the estimation techniques are equal the value for the distance is *0*. If it are different this value is *1*.

Development Market

The *Development Market* property stands for the target market of a project and has three possible options: *Inhouse Development*, *Customer Development* and *Anonymous Market*. To calculate the distance between development markets it was analyzed for *whom* the project is developed, what is the *main factor* for the *budget*, how *high* the *risk* of development is and how *predictable profit* is.
The *Inhouse Development* is for developing projects inside the company. They usually have the same procedure as other projects despite the fact that they don't yield a profit and have a low risk of development and a fixed budget. In the *Customer Development* the risk depends on the customer needs but are predictable. Profit is to be expected high and is connected with the project budget. In *Anonymous Market* developments a high risk is to be expected. Regardless of market analysis it is unsure that the project will be a success and generate profit. The budget for this market type is set from the company itself.
All distances between these markets are described in table 3.3.4.1 and are based on the above description. For example, the development for a customer is quite similar to an anonymous market development, so the distance between those is set to *one*. The big difference is that the profit is unsure for the anonymous market development. When comparing similar markets to each other the distance is *zero* as they are the same.

Development Market	Compared to	Distance
Inhouse Development	Customer Development	2
Customer Development	Anonymous Market	1
Anonymous Market	Inhouse Development	3

Table 3.3.4.1. Development Market - Distance

Development Type

Different *Development Types* describe the main reason for the project. This property contains the three types *New Project*, *Extension Project* and *Research Project*.

A *New Project* is developed if there is no previous project to build on or is a new conceptual formulation with no experiences so far. An *Extension Project* is made for expansion of existing software from previous projects. The third type are *Research Projects*, which are usually made to test an idea or to create a prototype for an analysis. These development types can be differentiated in the *main goal* of the project, *customer* and *profit level*. To introduce this differentiation the profit level describes the expected money and is divided into values from *zero* to *two*. *Zero* declines that the project does not generate any profit. The value *one* expects medium profit and *two* expects a high profit for the project.
The distances described in table 3.3.4.2 are based on the above descriptions. A comparison between *New Projects* and *Research Project* has the distance *two*. *Research Projects* do not generate any profit and have a different target than a *New Project*. The same applies with the comparison of an *Extension Project* with a *Research Project*. By comparing a *New Project* to an *Extension Project* the distance is set to *one* because the expected profit is smaller than in a *New Project*. The reason for this is that an *Extension Project* builds up on a previous project and is an addition to a previous contract which normally get a smaller budget.

Development Type	Compared to	Distance
New Project	Research Project	2
Extension Project	New Project	1
Research Project	Extension Project	2

Table 3.3.4.2. Development Market - Distance

Procedure Model

The next property is the used *procedure model*. There are *six* procedure models for selection available at the moment. The distance in table 3.3.4.3 is calculated by differentiating each procedure by *model type*, *suitable team size*, *formalization*, *industry focus* and *process cover*. *Model type* simply splits the procedure models in *sequential* and *iterative* types. The *suitable team size* indicates the laid out size of team members are useful for this model, which means with how much people this model can be used effectively. *Formalization* indicates how much effort has to be put into the documentation of the process steps. *Process cover* describes if the procedure model involves all main project processes, which are *development*, *test project management*, *quality assurance* and *change management* [MU10].
For example the *V-Model* is a sequential model for small and large teams. It is under a strong formalization pressure and covers all main project processes. It is used in all industry sectors and is strongly represented in sectors with high standards and formalization requirements such as the pharmacy industry. *Scrum* is an iterative model which is used for small teams. It does not cover all processes. Testing and quality assurance are not part of the scrum process. Scrum has no specific

industry sector and is represented in different industries. With this differentiation the distance is calculated as *5*, which is also described in table 3.3.4.3. The other distances are calculated the same way. A comparison between all available procedure models can be found in the appendix C.

Procedure Model	Compared to	Distance
Scrum	V-Model	5
Waterfall Model	V-Model	3
Spiral Model	V-Model	4
Iterative Development	V-Model	4
Prototyping	V-Model	6

Table 3.3.4.3. V-Model - Distance

Platform

Platform is the property that describes where the implemented software is deployed. The difference between platforms is calculated with the *porting costs* from one platform to another. This costs are calculated with the *main programming language* of the platform, *quality assurance*, *operating system family* and the *deployment difficulty*. Main programming language is the language that has the biggest portion by developing software for a platform. Quality assurance means the effort that has to be done for testing the developed project for a platform and *OS family* describes the family on which the platform is based on. The deployment difficulty describes the effort that has to be taken for projects to be installed.
For example the main programming language for *Android* is *Java*. Quality assurance has to be done with much effort as there are many different devices that support Android and is a member of *Unix-like* operating systems. The deployment effort is very low. An *Android* application is packed as as *apk*-file which will be installed automatically on a device. For *iOS* the main programming languages are *Objective-C* and *Swift*. The selection of devices is limited but the quality assurance is although at high effort because *Apple* has high restrictions for software. It also belongs to the *Unix-like* operating systems and it is more difficult to deploy software to devices. It is needed to have an developer account for testing software on the device and it is not possible to install software without the *App-Store*. As a result the distance between *Android* and *iOS* is set to *four*. This distance and the distance of Android to all other available platforms is described in table 3.3.4.4.

Industry Sector

The distance for *industry sectors* is not calculated with an equation as it is mostly an empiric property. It simply compares if the selected sectors are equal or not. If the industry sector does not apply to one of the available sectors there is the option to select *Other* as sector. For the calculation the distance between industry sectors is set to *zero* if they are equal and *one* otherwise. The available options for this property are:

Platform	Porting to	Costs
Android	iOS	4
Android	Windows Phone	3
Android	Mac OS	5
Android	Linux	4
Android	Windows 7	5
Android	Windows 8	5
Android	Windows 10	5
Android	Windows Universal App	6
Android	Web Development	5
Android	Other	7

Table 3.3.4.4. Platform - Porting cost

- *Agriculture*
- *Automotive*
- *Banking*
- *Bars and Restaurants*
- *Business Service*
- *Construction*
- *Education*
- *Electronics*
- *Entertainment*
- *Finance*
- *Health*
- *Internet*
- *Music Production*

Programming Language

The *Programming Language* describes the main language for the project. As an example for the platform Android the main programming language is *Java*. This Property has the most values for calculating the distance between two options. It is divided in two parts. **Part one** is calculating distance on the basis of *programming language properties* as described by Ruknet [RU95]. He categorized the programming languages with 12 different properties:

1. **Imperative**
   This value focuses on describing how a program operates. If it simply exists on a sequence of instruction for example.
2. **Object Oriented**
   This Property describes if the language supports the concept of objects.
3. **Functional**
   These property is a programming paradigm that treats computation as the evaluation of mathematical functions.

4. **Procedural**
   This programming paradigm allows routines and subroutines and contains a series of computational steps.
5. **Generic**
   Generic programming is a style which allows algorithms to be written in terms of types *to-be-specified-later* that are *instantiated* when they are needed.
6. **Reflective**
   This is the ability of a programming language to examine and modify its own structure and behavior.
7. **Event-Driven**
   An event-driven programming language is determined by events such as user actions or messages from other programs.
8. **Failsafe**
   The ability to throw exceptions if an operation or other system calls fails.
9. **Type Safety**
   States if a programming language discourages or prevents type errors which are stated in the values *safe* and *unsafe*.
10. **Type Expression**
    This property describes if the programmer must explicitly associate each variable with a particular type.
11. **Type Compatibility**
    This implies the use of a type checker in the programming language which verifies if an expression is consistent with the expected type by the context in which that expression appears.
12. **Type Checking**
    A process of verifying and enforcing the constraints of types at run-time or compile time.

Each property is converted to a numerical representation form which represents each of the possible values. For most of the properties this is simple, as the possible values are *true* or *false* which are represented as *1* and *0* in table 3.3.4.5. *Four* values can not be distinguished in true or false. For failsafe, these values are *safe*, *unsafe* and *unknown*. In the case of *failsafe* the value *0* means unknown, *1* is unsafe and *2* means safe. Type expression has also three values which are *implicit*, *explicit* an unknown. Implicit is represented as *1*, explicit as *2* and unknown as *0*. The three values for type compatibility are *nominal*, *structural* and *not stated*. These values are also represented as 1 for nominal, 2 for structural and 0 for not stated. The last property with more options is type checking. It can separated in *static*, *dynamic* and *unknown*. These are represented as *1* for *static* and *2* for *dynamic*. *Unknown* is as in the properties before represented as *0*.
This allows a calculation of the *Distance* of two programming languages. As this is calculated with the absolute value of each category. The constant c is the amount of all categories, which is *twelve*. For each category K_i of the compared programming language, the value of the other language is subtracted. This summed up is the

distance of these two languages.

$$Distance = \sum_{i=1}^{c} |K_i - K_i'| \tag{3.2}$$

Table 3.3.4.5 describes the comparison between *C++* and *Java* as an example. Those two programming languages only differ in *two* categories, namely reflective and failsafe. To calculate the distance for those two languages the difference between these categories is calculated with the equation from 3.2. The result is a distance of 3.
The **second part** is calculating conversion cost in another language which is the estimated effort for converting the source code. These are calculated with the equation 3.3, which estimates the cost factor for a conversion. These costs are simplified as the *Amount* of different categories plus an constant effort factor of *one*. This equation should illustrate that each different category generates effort for transforming the source code. It has to be adapted for different code paradigms and functionalities.

Programming Language	imperative	object oriented	functional	procedural	generic	reflective	event driven	failsafe	type safety	type expression	type compatability	type checking
C++	1	1	1	1	1	0	0	0	1	2	1	1
Java	1	1	1	1	1	1	0	2	1	2	1	1

Table 3.3.4.5. Programming Language - Distance

$$Conversion\ Costs = Amount + 1 \tag{3.3}$$

In the case of a conversion between *C++* an *Java* we have an amount of different columns of *2*. Summed with the constant cost factor from the equation the conversion costs, as described in table 3.3.4.6, are *3*.

Programming Language	Converting to	Distance
Java	Cpp	3

Table 3.3.4.6. Programming Language - Conversion Cost

To get a total distance between two programming languages, the distance and

conversion costs are summarized, as described in equation 3.4. In the example of *C++* and *Java* we have a distance of *3* and conversion costs of *3* which results in a total distance of *6*.

$$Total\ Distance = Distance + Conversion\ Costs \tag{3.4}$$

The available languages for this property are *C*, *C#*, *C++*, *COBOL*, *Clojure*, *Java*, *Javascript*, *Matlab*, *Objective-C*, *PHP*, *Prolog*, *Python* and *Scala*. The complete categorization and converting costs of all programming languages can be found in the appendix C.

Software Architecture

Software Architecture describes the high level structure of a software system and includes *software elements*, *relations* among them, and *properties* of both elements and relations. This property set the architecture of the project. It allows a differentiation on the fundamental structural choices which is important as it is costly to change a once implemented architecture.
The available options are *client-server*, *event driven*, *layered*, *microkernel*, *microservices*, *model view controller*, *serviceoriented* and *space based architecture*. These architectures are categorized with the help of *architecture patterns* as described by Richards [RIC15]. He analyzed software architectures by classifying them into *overall agility*, *ease of deployment*, *testability*, *performance*, *scalability* and *ease of development*. These are the categories in which a software architecture can be divided into. Each of the values is stated with *low*, *medium* or *high*, which are represented in the table and database as *1*, *2* or *3*.

- **Overall Agility**
  Describes the ability to respond to a constantly changing environment.
- **Ease of Deployment**
  Describes the deployment process and how changes affect deployment.
- **Testability**
  The way of how the architecture can be tested.
- **Performance**
  The ability a pattern can be lend to high-performance applications.
- **Scalability**
  Scalability describes how good the application with this pattern can be scaled.
- **Ease of Development**
  This describes the complexity for implementing the architecture.

Each software architecture can be differentiated with these properties. Another distinguishing feature is the *category*. Possible values are *distributed*, *interactive*, *adaptive* and *mud-to-structure*. The category is distinguished with an equality verification. Table 3.3.4.7 describes the categorization of *layered* and *eventdriven*

pattern as an example. The categorization of all available patterns can be found in the appendix C.
The *layered architecture* is the de facto standard for most *Java* applications and therefore is widely known by most *architects*, *designers*, and *developers*. It is time-consuming to make changes in this architecture pattern because of the monolithic nature of most implementations and one small change to a component can require a redeployment of the entire application. The pattern can be easily tested by mocking or stubbing layers. As some layer applications may run well, the pattern itself is not a high-performance architecture, also scaling applications with this pattern is hard due to different layers. Because this pattern is well known and not overly complex to implement the ease of development is considered as high.
The *eventdriven architecture* is a distributed asynchronous architecture pattern used to produce high scalable applications. Changes are generally isolated to one or a few event processors and can be made quickly without impacting other components. It tends to be easier in this architecture to deploy than the mediator topology, primarily because the event mediator component is tightly coupled to the event processors. Testing is complicated because of the asynchronous nature of this pattern. The pattern achieves high performance through its asynchronous capabilities and scalability is naturally achieved in this pattern through highly independent and decoupled event processors. Development can be complicated due to the asynchronous nature of the pattern as well as contract creation and the need for more advanced error handling conditions. The properties of the described patterns are visualized in table 3.3.4.7.
To calculate the distance between these two architectures, each property of the architecture will be subtracted with the property from the other architecture as described in equation 3.5. This calculation is the same as the distance for the programming languages. For the different categories the value *one* is added if they are not equal and *zero* if they are.

$$\textit{Architecture Distance} = \sum_{i=1}^{c} |P_i - P_i'| \tag{3.5}$$

In the case of the comparison of *layered architecture* and *eventdriven architecture*, as described in 3.3.4.7, the result of the distance calculation is *12*. As the categories are not equal the value *one* is added to the distance which results in a total distance of **13** for these two patterns.

Determination of the Total Distance

After determining the distance for each property the next step is calculating the *total distance* of two projects with the equation 3.6. Each property has to be multiplied with the weight from table 3.3.4.8, which sets distances with lower value the same importance as high distances. It also allows a prioritization of the different values.

Software Architecture	Category	overall agility	ease of deployment	testability	performance	scalability	ease of development
Layered Architecture	distributed	1	1	3	1	1	3
Eventdriven Architecture	interactive	3	3	1	3	3	1

Table 3.3.4.7. Software Architecture - Distance

$$Total\ Distance = \sum_{i=1}^{n} (D_i \cdot Weight_i) \tag{3.6}$$

The max distance of the programming languages is *28*, for example, whereas the highest value of the development type distance is *2*. To set more importance in the development type value this property is multiplied with *3*.

Property	Weight
Estimation Method	1
Development Market	2
Development Type	3
Procedure Model	2
Platform	1.5
Industry Sector	1
Programming Language	1
Software Architecture	1

Table 3.3.4.8. Property Weights

Calculation of the Relation

The last step is calculating the relation of projects with the *Total Distance* from equation 3.6. As the distance of the properties does not describe the relation but the difference between two projects, this is calculated with the equation 3.7. The *Max* value is the maximum possible total distance plus an addition of *10*. This should assure that two projects are never totally equal, because if they are they are the same and there is no need for creating a new project. This value describes the minimum difference of projects as 10%. The maximum distance between projects is *93*, so the *Max* value for the calculation is *103*. To get the relation this *Percentage Difference* has to be subtracted from 100%. The result of this equation is the *Percentage Relation* of two projects and allows to find related estimation to give

the user an overview of previous projects.

$$Percentage\ Relation = 100 - (\frac{Total\ Distance}{Max}) \cdot 100 \tag{3.7}$$

Example Calculation

To illustrate this calculation this sections describes a sample calculation with the project properties from table 3.3.4.9. As described above, the first step is calculating the distance of each property. These distances calculated with the equation 3.6 and the weights from table 3.3.4.8 result into a total distance of *23.5*. Next step is to calculate the percentage relation with equation 3.7, which results into a relation of **77.18%** of both projects.

	Project A	**Project B**	**Distance**
Estimation Technique	Function Point	Function Point	0
Development Market	Anonymous Market	Inhouse Development	3
Development Type	New Project	New Project	0
Procedure Model	Scrum	Scrum	0
Platform	Android	Windows Phone	3
Industry Sector	Education	Internet	1
Programming Language	Java	C++	6
Software Architecture	Model View Controller	Client-Server	4

Table 3.3.4.9. Example Project Properties

3.3.5 Project Analyzer

The task of this component is to show how the estimations changed over time. It should show each project with the *estimated days* and the *final days*. Depending on the estimation technique the analysis should show the amount of the estimation. This analysis is visualized in diagram 3.6. It should help visualizing the progress of the estimation. As with further estimations, the gap between the estimated days and the final days should become smaller. If the diagram shows project with a high gap between the final man days and the estimated man days it means that the estimation was not precise enough and it has to be analyzed what went wrong.

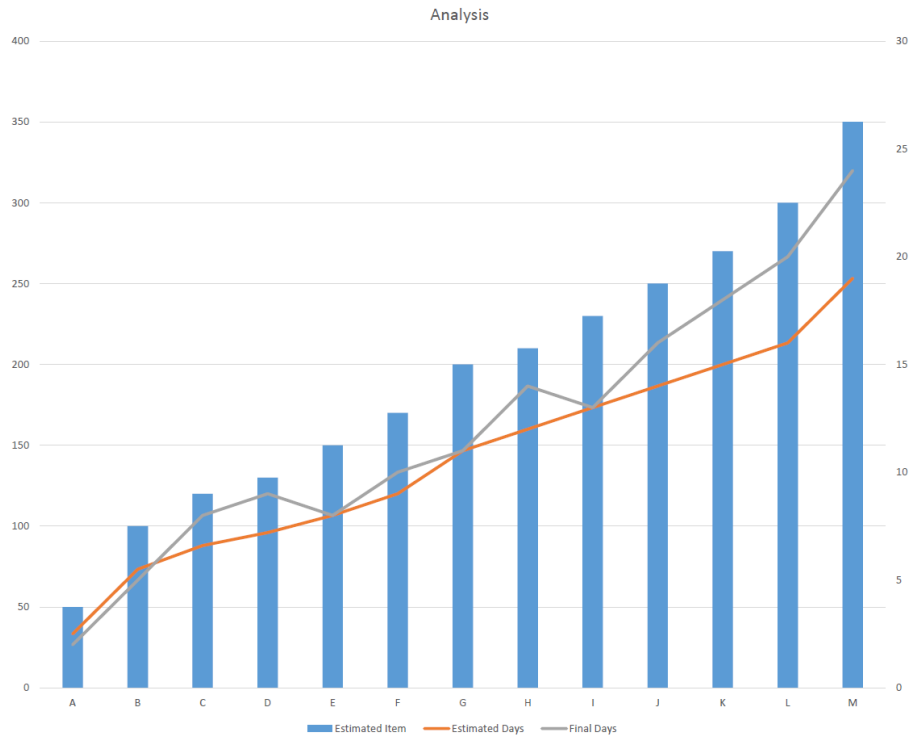

Fig. 3.6. - Analysis Diagram

3.3.6 Minor Components

Beside the main components their are some minor components which are not essential but are mentioned here.

Export

The *Export* component is responsible for preparing a project as a table which should allow further processing of the estimation. All items of the estimation will be transformed to an `csv` file according to the estimation technique. Also all properties

of the project will be exported to the file. This allows *printing* the estimation or sending it via *e-mail*.

Accessing Resources

To implement a multilingual option for data from the database this component should do the essential work. All values of data strings are stored in the database as `'<NAME>_string'`. This value is also stored in the string resource file of Android and is connected to the string value in the device language. These files already support multilingualism which means that this component only needs to build a connection between the *resource files* and the *database value*.
If a value from the database is queried, the component will look for the name in the resource files. The right value in the device language will then be returned as a string. To add a string id from the database to a new entry the process works exactly the same in the other direction. The component searches for the belonging id of a string. The id is the representation of resources in Android which are newly generated after each start of the application and have to be managed in the component. This *id* is linked to the database string value and allows queries of the *Database Helper* component.

Project Filter

The Project Filter helps to maintain an overview of all projects. It is an addition to the *Project Overview* which allows restriction of the displayed projects. Deleted projects will not be displayed by default, which can be set with the filter. Another possible option is to display only projects of a specific *Estimation Technique*. This component is held modular, so it is simple to add new parameters for filtering the projects.

Help Articles

The *Help Articles* allow the user to inform them self about the functionalities of the application and fundamentals like *Function Point*. This should be a guide for new and unexperienced users. All Articles are stored in an `XML` file which will be synchronized with the server in a later version of the application. The component is responsible for reading all articles from this file and allow the application to display them. These articles are not stored in the database as if they may have any desired length and must be available in different languages.

Project Analyzer

This component evaluates all project data and delivers an overview in form of diagrams. Figure 3.7 is an example of such an overview. This statistic shows how many projects used which programming language as their property. Purpose of this is to help users an overview of how often properties are used. This component allows statistics of all properties and about the relation of *active* and *terminated* projects.

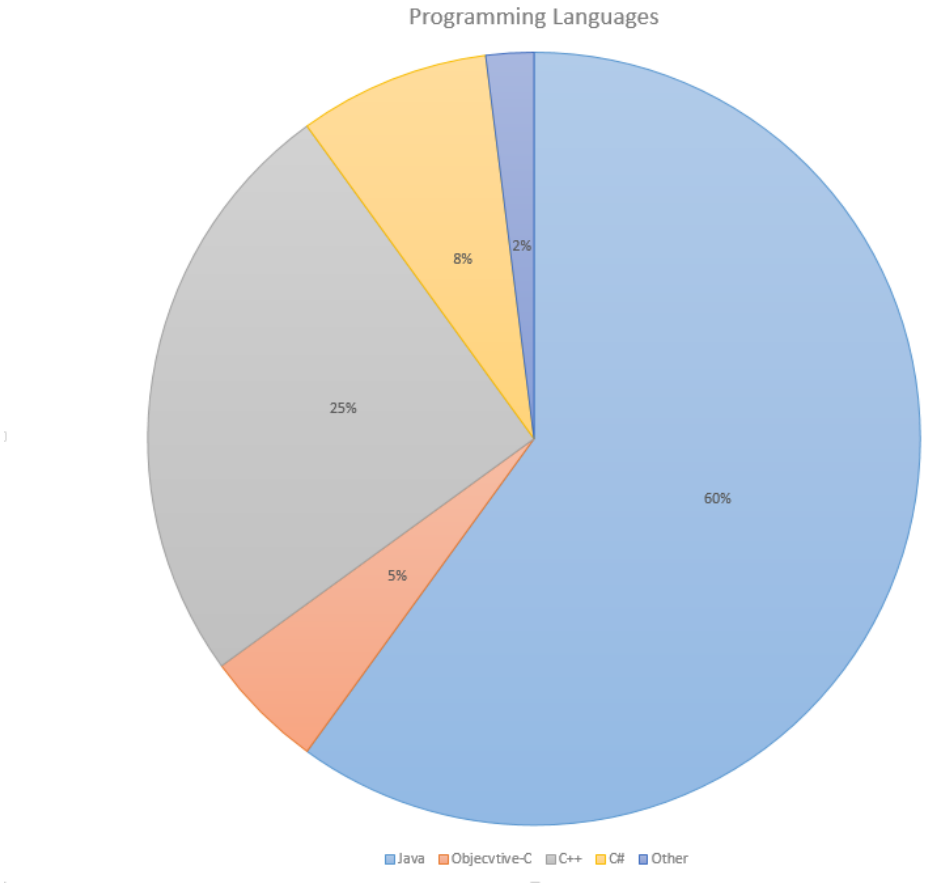

Fig. 3.7. - Use of Programming Languages

3.4 Database design

This section introduces the design of the databases. To allow using a filled database in the application, without creating it with *SQL* commands during the first start, the database was designed and filled before the implementation. The process of accessing an existing database is described in section 4.3. There are *two* separated databases in the application. The *Project Database*, which inherits all informations about projects, estimations and properties, and the *Userinformation Database*, which is designed for a further use with a server and inherits user informations and server messages.

3.4.1 Project Database

The *Project Database* is the main storage component in the application. It contains all important informations for the *Project Relation Solver* and possible values for selecting *Estimation Method* and *Properties*. All created projects and influence factors will be stored here and allow a consistent processing. All tables must have a primary key column which is called *_id* or otherwise this table is not recognized

by Android. Also the table *android_metadata* is needed in the database. This table contains only one column called *locale* and inherits the standard localization of the database, for example `en_US` for the American localization.

Project Properties

All tables for available properties have the same structure. They contain the column *_id* and *name* as described in figure 3.8. The *_id* is primary key of each property value and also foreign key in each table where a property is needed. Storing the name of a property is as described in section 3.3.6, done by writing the name of a property suffixed with '_string'. This is done in the column *name*.

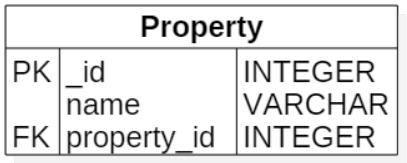

Fig. 3.8. - ER Diagram Project Property

Figure 3.9 describes the *ER diagram* for the *programming language* property concluding the tables for conversion costs and language properties. This is described in this section as an example of properties. The complete entity relation diagram can be found in the appendix D. As described above the table *ProgrammingLanguage* contains an *_id* column and a column with the *name* of the programming language. The column *property_id* is a foreign key and is a reference to the table *ProgrammingLanguageProperty*. This table contains the properties of a programming language as described in section 3.3.4. The table *ProgrammingLanguageConversionCosts* contains the previous calculated costs for converting source code from one programming language to another.

Projects

The data of projects is stored in several tables. As described in figure 3.10 these tables are named *Project*, *ProjectDetails*, *EstimationTechnique* and *ProjectProperty*. Also the tables *FunctionPointInfluenceFactor*, *FunctionPointEstimation* and the *estimation item* tables, here portrayed as the table *InputData* are important for an estimation with function point.
The *EstimationTechnique* table stores all available techniques for projects. It contains only the *id* and *name* of a technique. The *Project* table itself contains the selected estimation technique as a foreign key. As described in fig. 3.10, the connection between *Estimation Technique* and *Project* is a *many-to-one* connection. A

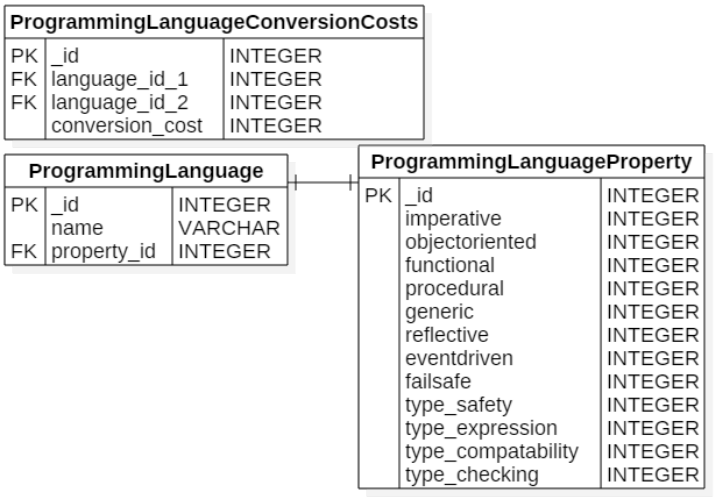

Fig. 3.9. - ER Diagram Project Property

project can only have one estimation technique, but one *Estimation Technique* can be used by many projects. Further columns are the *name* of the project and the *creation date* in the column *created_at*. The column *is_deleted* is a flag to state if the project was already deleted. This allows *restoration* of a project. Also the column *is_terminated* stands for the termination of a project. The last column is the *project_detail_id* which is also a foreign key to the table *ProjectDetails*. The connection here is a *one-to-one* relation. This table mainly contains foreign keys to other tables but for the estimation and influence factors it is not possible to set these foreign keys. This depends on the selected estimation technique and selection of the right table is made in the implementation. The column *evaluated_person_days* stores the amount of days the estimation technique calculated for a project. To load the right project icon the column *icon_id* refers to the table *ProjectIcons*, which stores *name* and *path* of an icon. How this icon is loaded is described in section 4.6. As descriptions can be a long `string`, they are stored in an extra table called *ProjectDescription*. This is made for a better overview of the *ProjectDetails* table. The last foreign key in *ProjectDetails* refers to the table *ProjectProperty*. It contains the id of selected properties for a project.
The estimation itself depends on the selected estimation technique. In this paper only the *Function Point* technique is implemented and described. The estimation itself is stored within the *FunctionPointEstimation* table. It contains foreign keys to the estimation items for each category. Each of this tables is structured the same as the described *InputData* table. It contains the *_id*, *simple*, *medium* and *complex* value of a category. The other important table is *FunctionPointInfluenceFactor*. All influence factors as described in section 2.2.1 are stored here.

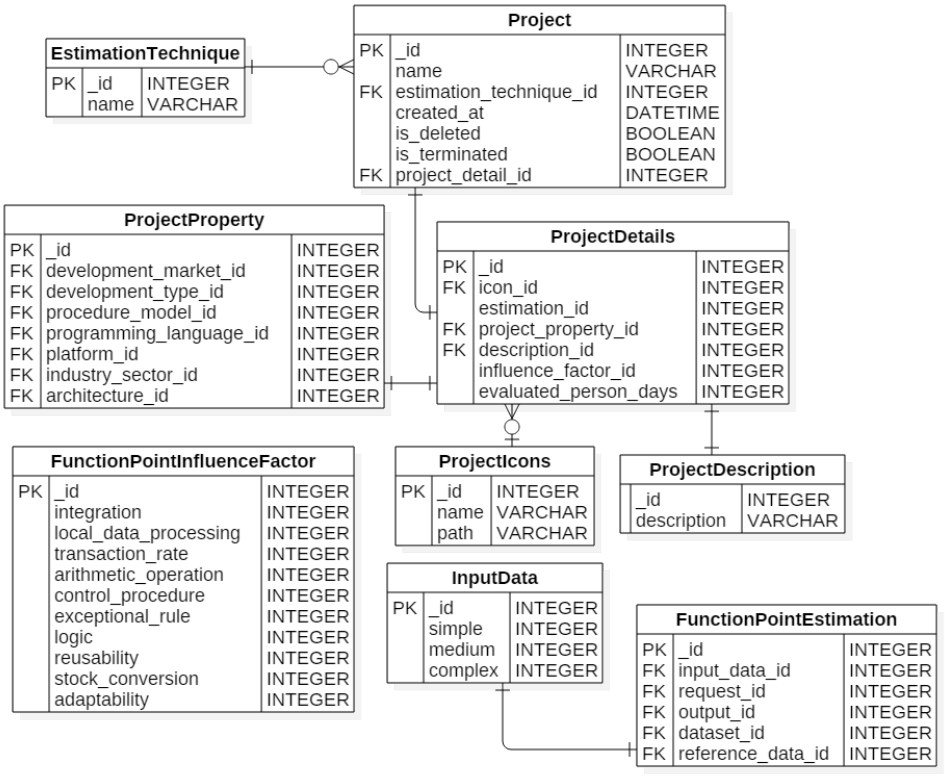

Fig. 3.10. - Entity Relation Diagram for Project Informations

3.4.2 Userinformation Database

This database was designed for future use of a server connection. The *User* table stores the information of an authorized user on the device. It contains *name*, *password* and *mail* address to identify a user. Also the column *device_id* gives information of the device he is using. The *MessageQueue* table stores all messages that where synchronized from the server. This contains for example updates for the project properties.

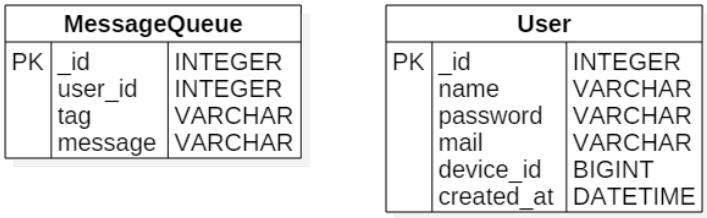

Fig. 3.11. - Entity Relation Diagram for User Informations

3.5 User Interface

This chapter describes the user interface design of the main screens and when which element is used. Each screen is designed with the design guidelines from *Android* and the *Material Design* rules [GO16].
Most data is displayed in lists, which allows to add new content during runtime without limitation of the size. The other way data is displayed are the so called `Spinner`, which are an expandable drop down list. This list can be filled with values from databases during runtime. Important functions that represent main functionalities of the screen are displayed as a `Flow Button`. This type of a button *'float'* above the layout and is located on the bottom right of the relevant screen. Other important functions are displayed as a toolbar option. These are only represented with icons to save space. Secondary functions are available in option menus, which are indicated as three dots in the toolbar or as a menu drawer. Functionalities that operate directly upon an item are connected to this. Those functions can be invoked with a *click* on that item and a menu will show up.

3.5.1 Projects Overview

Project Overview is the first screen the user will see when starting the application. All active projects are visible in a list, as shown in figure 3.12. These projects are listed there with their *name*, *estimation technique*, *creation date* and the *project icon*. This list is scrollable and inherits all projects that are not deleted. The circle with the *'+'* on the bottom right will open the creation of a new project. On the upper right are the functions to start the *search* and open the *project filter*. The main menu of the application is on the upper left corner. This is a *drawer* menu which will be visible after a click. Therefore are the additional options and screens only visible if the user needs them and don't use space of the screen.

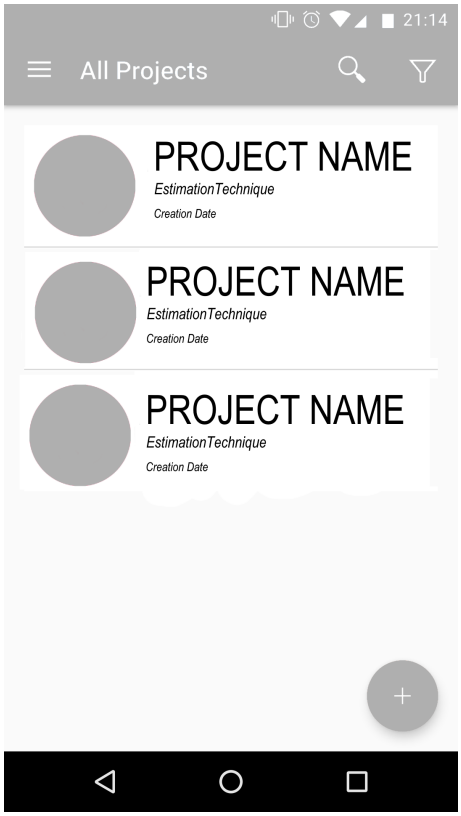

Fig. 3.12. - Project Overview

3.5.2 Project Creation

The GUI design in figure 3.13 describes the screen for creating projects. This creation is done in *six* different screens, which are not shown here. All of them have the same design. The circles on the bottom indicate the progress of the creation. White a *swipe right* or *left* the user can swap through the screens. All informations are saved automatically. Figure 3.14 shows the list overview of the creation. This screen serves for the purpose to have all options in one view and check the input. With the `'SAVE PROJECT'` function in the upper right corner the project will be saved and the *Project Overview* screen will show up.

3.5.3 Estimate Function Point Project

The *Function Point Estimation* screen is shown in figure 3.15 and 3.16. The estimation is separated into two tabs. The first tab all estimation items which can be edited with the *pen* symbol. It shows also the *Total Points*, *Evaluated Function Points* and *Evaluated Person Days*. This attempts to only view the essential informations of the estimation. Each of the fields displays the actual calculated value. All values for each estimation category are the *calculated points*, which means *amount of category* multiplied with the *weight*. The other tab, as showed in 3.16,

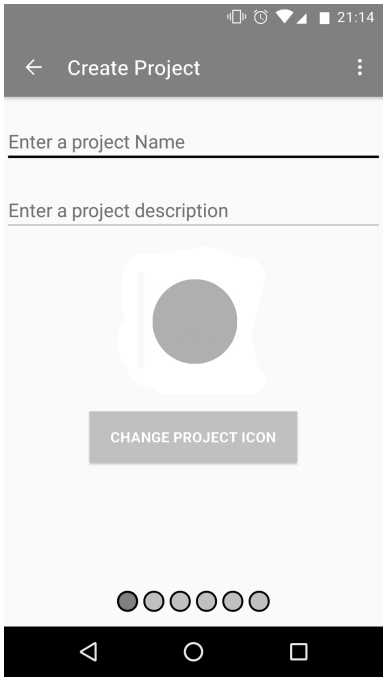

Fig. 3.13. - Project Creation

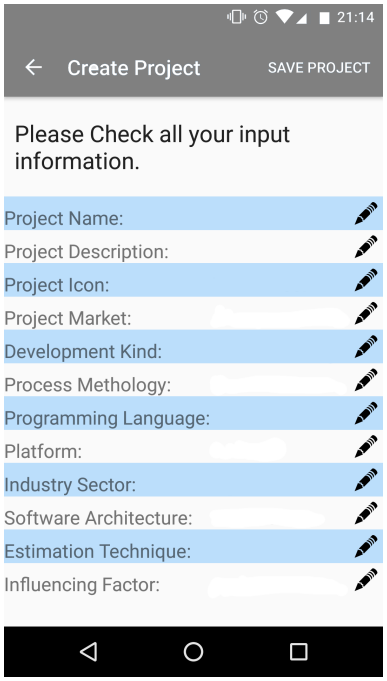

Fig. 3.14. - Creation Overview

inherits the influence factor, which is displayed in the list. It shows the chosen value for each influence option. Also the sum of the influence factor is displayed as well as the calculated rating. With the `'CHANGE FACTOR SET'` button the user can choose from all available influence factors.

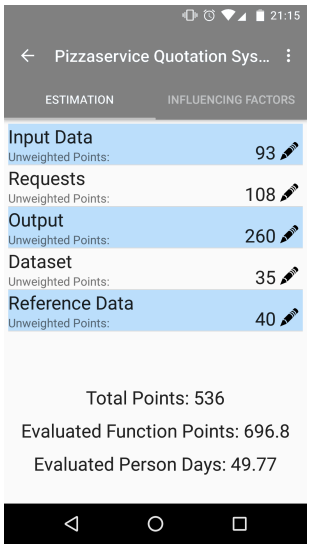

Fig. 3.15. - Function Point Estimation

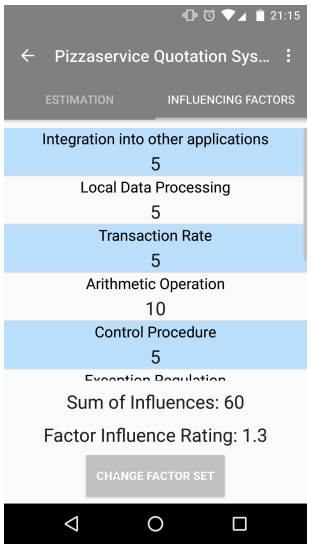

Fig. 3.16. - Function Point Influence Factor

4

Implementation

The used libraries and selected implementation approaches are described in this chapter. *Android Studio* was used as the *IDE* for development and *GitHub* was used for version management but are not described in this paper.

4.1 Used Libraries

For the implementation were *two* external libraries necessary. These libraries are called *POI* and *mpandroidchartlibrary*. *POI* is a library from the *APACHE Software Foundation* and is open source. It allows a conversion of data into Microsoft documents. This library is used in the *Export* component and allows to create `xls` documents with the project data. The *mpandroidchartlibrary* library is also an open source library developed by Philipp Jahoda. This library provides many different diagram types for *Android* such as *pie* and *bar charts*. This charts are used for the estimation technique analysis and for the project statistics.

4.2 Project Structure

Android divides the project into an *assets*, *resources* and *java* folder. The *assets* folder stores project icons and the *two* databases of the project. After installation of the `apk`, these files are moved to the internal folder space of the application. The *resources* folder contains every `layout` file, `string` resources and the android `icons`. All resource files are typically `XML` files or in case of images `png` files. The *java* folder contains the complete source code. This folder was divided into the sub folders *Activities*, *Fragments*, *DataObjects*, *Server* and *Util*. The folders *Activities* and *Fragments* contain the associated classes for each layout and are responsible to control the user input. In the folder *DataObject* are all data classes stored. These are objects that only store data and don't have any logical components. The *Server* folder contains at the moment only the `ServerConnector` class. This folder was created to separate the future server connection. Last folder is the *Util* folder. It contains classes that process data, for example the `DatabaseHelper` class.

4.3 Pre-filled Database

The advantage of a pre-filled database is a shorter initialization after first start of the application and the possibility to deploy content. The database doesn't have to run many *SQL* scripts to create tables and insert values. Also values and tables can be created with an external tool such as *SqliteBrowser*[1]. This allows testing of the *tables* and *queries* that are used in the application. To use this database in the application two steps are essential:

1. **Create Database**
2. **Open Database**

Both methods are inherited in the class `DataBaseHelper`. Listing 4.1 shows the method `createDataBase()` which is responsible for the creation of the database. At first the method checks with `checkDataBase()` if the database does already exist in the `application` folder. If it does not exist the method calls `copyDatabase()` which will read the database from the `assets` folder and write a new database to the application folder.

```
1  public void createDataBase() throws IOException
2  {
3     boolean dbExist = checkDataBase();
4     if (!dbExist)
5     {
6        this.getReadableDatabase();
7        try
8        {
9           copyDataBase();
10       } catch (IOException e)
11       {
12          Log.d("ERROR", "Database could not be copied " + e.getCause());
13          throw new Error("Database could not be copied");
14       }
15    }
16 }
```

Listing 4.1. Method *createDataBase*

Open the database is done with the method `openDataBase()`, which is described in listing 4.2. The class variable `projectEstimationDataBase` is initialized here with the `SQLiteDatabase` command `openDatabase` which create a `SQLiteDatabase` object with the path where it is stored. This allows the execution of `SQL` queries on this object.

```
1  public void openDataBase() throws SQLException
2  {
3     String myPath = DB_PATH + DB_NAME;
4     projectEstimationDataBase = SQLiteDatabase.openDatabase(myPath, null,
5                                 SQLiteDatabase.OPEN_READWRITE);
6  }
```

Listing 4.2. Method *openDataBase*

[1] `http://sqlitebrowser.org/`

4.4 The Database Helper

The `DataBaseHelper` class has the most lines of code in the project. As `SQL` queries can be generalized it is not possible to generalize the preprocessing of the data. Different date needs different treatment for the calling method. Each table has also various column names what is another difficulty for generalization.
All requests with `SELECT` statements have a similar structure. As described in listing 4.3, the first step is to build a `String query` with the *table name* and *parameter* from the method. Next step is to get a readable database from the `SQLiteDatabase` object of this class. The query will be executed on this object and creates a `Cursor`, which is the result table. To get the values from this `Cursor` each column has to be addressed and saved into a variable which will be returned to the caller. If more values are needed those have to be packed into other data structures.

```
1  public void selectSomethingFromTable(String id)
2  {
3     String query = String.format("SELECT _id FROM <TABLENAME> WHERE id = '%s'",
4                                  id);
5     SQLiteDatabase db = this.getReadableDatabase();
6     String name = "";
7     try (Cursor c = db.rawQuery(query, null))
8     {
9        if (c.moveToFirst())
10       {
11          name = c.getInt(c.getColumnIndex("<COLUMN-NAME>"));
12       }
13    }
14    db.close();
15    return id;
16 }
```

Listing 4.3. Method *openDataBase*

For the query types `DELETE` and `ALTER TABLE` a method needs to instantiate a writable database with `'SQLiteDatabase db = this.getWritableDatabase()'`. The `db` objects offers the possibility `update` to change existing values in a table. It gets an object `ContentValues` as a parameter which inherits the column *name* and *value* which has to be updated. The other to parameters for `update` are the table *name* and *selection* to find the right row. To delete a row the method `delete(String table, String whereClause, String[] whereArgs)` is needed. It has the parameters that indicate the table and which row should be deleted.

4.5 Multilingual Strings

As described in section 3.3.6 all non user edited names in the database are connected with the android `string.xml` resource. To achieve this connection a `HashMap` is needed which connects the resource names with the `id` *Android* assigns to each string. It is important that the name in the database is **exactly** the same as in the string `xml`, otherwise the string won't be found and causes an error. It is also important to assure that every string key is unique to avoid false string being

loaded. This map is called `resourcesIdMap` and has the name from the database as key and the associated resource id as value. Two methods that process this `HashMap`: `getStringResourceValueByResourceName (String resource Name)` and `getStringResourceNameByResourceValue (String resourceValue)`. The parameter `ResourceName` is the name that is described in the database and `ResourceValue` is the string from `strings.xml`.
Listing 4.4 shows the source code for reading the value of a resource with it's key. For example the key `'test_string` returns as a result the string `'test'`.

```
1  public String getStringResourceValueByResourceName(String resourceName)
2  {
3     int resID = context.getResources().getIdentifier(resourceName, "string",
4     context.getPackageName());
5     String name = context.getResources().getString(resID);
6     DataBaseHelper.resourcesIdMap.put(name, resID);
7     return name;
8  }
```

Listing 4.4. *getStringResourceValueByResourceName*

4.6 Load Project Icons

In the table `ProjectIcons` are the informations for all available icons stored. The important column for loading the icon is called `path`. It stores the *relative path* to the icon and the *icon name*. The method `loadProjectIcon` has to look if the icon exists at this position. As described in listing 4.5, this icon is decoded in a `Bitmap` and returned to the caller. `Android` can present `Bitmaps` without further effort.

```
1  public Bitmap loadProjectIcon(String path)
2  {
3     AssetManager assetManager = this.context.getAssets();
4     InputStream istr;
5     Bitmap projectIcon = null;
6     try
7     {
8        istr = assetManager.open(path);
9        projectIcon = BitmapFactory.decodeStream(istr);
10    } catch (IOException e)
11    {
12    }
13    return projectIcon;
14 }
```

Listing 4.5. Method *loadProjectIcon*

4.7 Calculate Person Days

This functionality is at the moment only adapted to the *Function Point* technique. For calculating the *person days* or *man days* two methods are necessary. One is the `evaluateFunctionPointPersonDaysWithBaseProductivity`, when there are no terminated projects to calculate a *points-per-day* value. The other method, `evaluateFunctionPointPersonDaysWithExistingProductivity`, is called when

there are already terminated projects which will help get the best fitting *points-per-day* value. The calculation with the base productivity checks the database for the range of the total points of the project. If a project hast *total points* of *500*, for example, it fits in the base productivity that goes from *350* to *650* points, which has a *points-per-day* value of *16*. These *500* points will then be divided with *16* which results in *31.25 man days*.
If there are enough terminated projects the algorithm tries to calculate a more accurate *points-per-day* value. The calculation checks all terminated projects for a project with less total points and for a project with more total points than the calculated project. Within these values the algorithm searches for those projects which total points are nearest to the calculated project. If one of these items are empty the algorithm selects the base productivity for this value instead. Listing 4.6 shows the code that is executed afterwards. For the smaller and bigger item the average *points-per-day* are calculated. This value is then divided with the selected projects *evaluated points* which returns the *evaluated days* for the project.

```
1 double averagePointsPerDay = (smallerItem.getPointsPerDay() +
2             biggerItem.getPointsPerDay()) / 2;
3 averagePointsPerDay = roundDoubleTwoDecimals(averagePointsPerDay);
4 evaluatedDays = roundDoubleTwoDecimals(project.getEvaluatedPoints()
5             / averagePointsPerDay);
```

Listing 4.6. Evaluate Days

4.8 Find related Projects

The relation of two projects is calculated in the `ProjectRelationSolver` class. It is initialized with all projects that are listed in the database and inherit *active* and *deleted projects*. Also the selected project for comparison is a parameter at initialization. The main method for finding related projects is `getRelatedProjects` and has the `relationBorder` as parameter which defines the percentage border for relation. It calls for each project the method `compareWithChosenProject` which compares a project with the selected one. As most of the property distances are save in the database, this method only needs to select the right distance from the database. Listing 4.7 shows this selection with the development market distance as example. It calls the method `getPropertyDistance` from the `DatabaseHelper`. It gets the table name of the property and the table name of the distance as parameter. Also the names of the columns for selecting the right value. The last two parameters are the values that have to be compared.

```
1 double developmentMarketDistance = databaseHelper.getPropertyDistance
2           ("DevelopmentMarkets", "DevelopmentMarketComparison",
3           "market_id_1", "market_id_2", "comparison_distance",
4           project.getProjectProperties().getMarket(),
5            p.getProjectProperties().getMarket());
```

Listing 4.7. Development Market Distance

For `software architecture` and `programming language` these distance has to be calculated. This is done by loading the properties from the database and sub-

tracting them. These distances are then added to the `distanceSum` which is then calculated with the equation in listing 4.8. For logging purposes the calculation was separated into two parts. Before returning the `differencePercentage` it has to be subtracted from *100* to get the relation.

```
differencePercentage = databaseHelper.roundDoubleTwoDecimals(distanceSum
                                / 103) * 100;
```

Listing 4.8. Percentage Difference Calculation

4.9 ListView Elements and ViewHolder

The difficulty with `ListViews` was to achieve that each item remember his content and can be processed. For example selecting a *button* in the list element which returns the name of the item. This was achieved by using so-called `ViewHolder`. These are sub-classes that inherit all `View` elements of the list item. This is done in the adapter of `ListViews`. The method `getView` is called for filling the `ListView` with items. One parameter of this method is the `convertView` which represents the actual item. The trick to achieve that each element remembers its content to add the `ViewHolder` as a tag to the `convertView`. Listing 4.9 shows an example of this allocation. For each *list item* a new `ViewHolder` is initialized. Each including `View` element has to be assigned with the element inside the `convertView`. Finally the `holder` is assigned as a tag to the `convertView`.

```
convertView = vi.inflate(R.layout.
    function_point_influence_factorset_with_subitems_list_item, null);
holder = new ViewHolder();
holder.tvShowSubitems = (TextView) convertView.findViewById(
    R.id.tvShowSubitems);
convertView.setTag(holder);
```

Listing 4.9. ListView Holder example

5

Software Test

To test the application test cases were created which rely on the requirements and should evaluate if the functionality is implemented in the application. The tests were performed by some fellow students which returned a first feedback for the application.

5.1 Test Cases

Test Cases describe an elementary, functional software test to review a specification. All tests contain of the following elements:

- *Prerequisite* (only if necessary)
- *Test object*
- *Input data*
- *Procedure*
- *Result*
- *Postcondition*

These describe the main functionality which has to be tested. The test document comprised all test cases and for each step an extra field where the result could be noted. These *Test Cases* can be found in the appendix E. The following *Test Case* is an example and describes the function to *delete a project*:

Delete Project

Prerequisite
The application shows the *Project Overview* screen and projects are visible.
Test object
One available project.
Input data
none
Procedure

1. `Longclick` on the chosen project.
2. A *popup* menu is shown on the screen.
3. Click on `'Delete Project'`.

4. Another *popup* shows on the screen to confirm the deletion.
5. Click on `'Yes'`.

Result
The project is deleted and will no longer be shown in the *Project Overview*
Postcondition
The project can be recovered in the database settings.
By processing these tests the users found some bugs and additions for the application which were resolved for the version which is attached at this thesis. As a result all functionalities which are implemented with this version work accurate.

6

Conclusion

This paper presented the possibility to process *cost estimation* with a mobile application. It tries to give an approach to make projects *comparable* and build a base for a mobile cost estimation application. It focuses on the implementation of *Function Point*. This estimation technique is completely workable. The estimation can also be exported as an excel sheet for further processing. The project was estimated as **14** *man days*. The Development itself was done in **19** *man days* which is a difference of the estimated time of **5** *man days*.
The developed application allows a *comparison* between projects which relies on *seven* different project properties. These are calculated with a distance which gives the *total difference* of two projects. Those project properties are implemented and allow a categorization and give traceable relation results. To use the application properly it is assumed that the user knows how the *estimation techniques* work. He has the possibility to use the *Help* inside the application but a basic understanding is required.
One of the biggest problems was to create a user interface which inherits all functions to estimate a project and also supports the *android design principles*. For the creation of the user interface many *ListViews* are used, which allow the possibility for quick and dynamic extension of these lists. These caused an extra effort for making them save all informations, but afterwards the use of *ListViews* is unproblematic. Another problem was to create a *algorithm* for comparing projects. It had to be evaluated which properties make sense and can be set for every project. These properties must also have relevance for the algorithm. To create this algorithm I had to develop a possibility to calculate a distance which can be transfered into a percentage difference.
As an outlook for further development many features and additions are possible. One addition is to implement more estimation techniques such as *COCOMO*. This would support more informations about how long a project can take because different technique have another effort as output. It has to be evaluated if as a consequent, it is needed to make a project transferable to other estimation techniques or if projects need a different structure which allows more estimation techniques. Another addition is to create a *server* which stores all estimations from the user. This would support more possibilities to improve the estimations and all users with access could rely on the informations of other users. This would constantly expand

the estimations and would give more precise results. It could also allow sharing of projects which gives other users the possibility to look up estimations in detail and adapt those informations for their project. A server management tool would also provide the possibility to manage the *master data* of the project comparison from the server. This would make it easy to create more options for the project properties. An *adaption* of the application for tablets would provide the possibility of a different user experience.
All in all it can be said, that the mobile application for cost estimations offers a *promising alternative* to the traditional estimation software, that is worth further *research* and *evaluation*. The application will be extended within the *'Projekt-studium'* for my *master degree* at the *University of Applied Science in Trier*. In the winter semester *2016/2017* this application will also be used in the module *'Spezifikation interaktiver Systeme'* which will give more feedback about this application and allows a field study.

References

AB14. Abubaker Ali, Noraini Ibrahim: *Comparative Analysis Between Fpa and Cocomo Techniques For Software Cost Estimation.* International Journal of Engineering Research and Development, 2014.

AL79. Albrecht, Allan J.: *Measuring Application Development Productivity.* IBM Corporation, 1979.

BA08. Balzert, Helmut : *Lehrbuch der Softwaretechnik: Softwaremanagement.* Spektrum Akademischer Verlag, 2008.

BA09. Balzert, Helmut : *Lehrbuch der Softwaretechnik: Basiskonzepte und Requirements Engineering.* Spektrum Akademischer Verlag, 2008.

BO81. Boehm, Barry: *Software Engineering Economics.* Englewood Cliffs, 1981.

CA04. Camarinha-Matos, Luis M.: *Evaluating a Software Costing Method Based on Software Features and Case Based Reasoning.* expert, 2004.

DU14. Dumslaff Uwe, et al: *Studie IT-Trends 2014.* Capgemini Deutschland Holding GmbH
`https://www.de.capgemini.com/resource-file-access/resource/pdf/capgemini-it-trends-studie-2014.pdf`. *(Stand: 04.02.2016)*

FI10. Fischer, Christian and Aier, Stephan: *IT-Projektkostenschätzung – Ein pragmatischer Ansatz.* Institut für Wirtschaftsinformatik, Universität St. Gallen, 2010.

FI16. Fischmann, Lee : *Evolving Function Points.* Galorath Inc.

GA11. Galorath Incorporated : *SEER for Software.* Galorath Incorporated , 2011.
`http://galorath.com/wp-content/uploads/2014/08/SEERforSoftware2.pdf` *(Stand: 04.02.2016)*

GE11. Geirhos, Matthias: *IT-Projektmanagement: Was wirklich funktioniert - und was nicht .* Galileo Computing, 2011.

GO16. Google: *Android Design Principles .* Google Inc.
`http://developer.android.com/design/get-started/principles.html`. *(Stand: 04.02.2016)*

HU04. Hürten, Robert: *Function-Point Analysis - Theorie und Praxis: Die Grundlage für das moderne Softwaremanagement.* expert, 2004.

HU11. Hummel, Oliver : *Aufwandsschätzungen in der Software- und Systementwicklung kompakt*. Spektrum Akademischer Verlag, 2011.

ISBSG10. ISBSG : *Practical Software Project Estimation: A Toolkit for Estimating Software Development and Duration*. Mcgraw-Hill Education - Europe, 2010

JE01. Jenny, Bruno: *Projektmanagement in der Wirtschaftsinformatik*. vdf Hochschulverlag AG an der ETH Zürich, 2001.

KA16. Kathleen, Peters: *Software Project Estimation*. i.c. stars, 2016.

LI07. Litke, Hans-Dieter: *Projektmanagement: Methoden, Techniken, Verhaltensweisen*. Carl Hanser Verlag GmbH & Co. KG, 2007.

LO14. Lopez, Carola : *Faszination Mobile - Verbreitung, Nutzungsmuster und Trends*. BVDW , 2014.
`http://www.bvdw.org/presseserver/studie_faszination_mobile/BVDW_Faszination_Mobile_2014.pdf` *(Stand: 04.02.2016)*

LO16. Longstreet David: *Fundamentals of Function Point Analysis* . softwaremetrics
`http://www.softwaremetrics.com/`. *(Stand: 04.02.2016)*

MC06. McConnell, Steve : *Software Estimation: Demystifying the Black Art)*. Microsoft Press, 2006.

ME16. Meyer, Dagmar : *Projektmanagement - Aufwandsschätzung*. Ostfalia - Hochschule für angewandte Wissenschaften.
`http://www.ostfalia.de/export/sites/default/de/pws/meyer/lehrveranstaltungen/ma_projektmanagement/projektmanagement_unterlagen/Folien_Aufwand.pdf`. *(Stand: 04.02.2016)*

MU10. Munassar, Nabil, A. Govardhan: *A Comparison Between Five Models Of Software Engineering* . Jawahrlal Nehru Technological Universit *(Stand: 04.02.2016)*

OM14. OMG: *Automated Function Points (AFP)*. OMG - Object Management Group *(Stand: 04.02.2016)*

PA10. Partsch Melmuth: *Requirements-Engineering systematisch*. examen.press, 2010.

PO12. Poensgen, Benjamin: *Function-Point-Analyse: Ein Praxishandbuch* . dpunkt.verlag, 2012.

PR15. PRICE Systems : *PRICE Systems - Overview*. PRICE Systems , 2015.
`http://www.pricesystems.com/en-us/leadership/priceoverview.aspx` *(Stand: 04.02.2016)*

RIC15. Richards, Mark: *Software Architecture Patterns*. O'Reilly, 2015.

RI15. Ripsas, Sven and Tröger, Steffen : *3. Deutscher Startup Monitor* . KPMG, 2015.
`http://deutscherstartupmonitor.de/fileadmin/dsm/dsm-15/studie_dsm_2015.pdf` *(Stand: 04.02.2016)*

RU95. Ruknet, Cezzar: *A Guide to Programming Languages: Overview and Comparison*. Artech House Publishers, 1995.

SO07. Sommerville, Ian: *Software Engineering.* Pearson, 2007.

ST14. The Standish Group: *The Standish Group Report.* Project Smart, 2014.

WE16. Weigand, Florian: *Aufwandsschsatzung in IT-Grosprojekten - Function Point Methode.* TU München.

WI05. Wieczorrek,Hans W: *Management von IT-Projekten. Von der Planung zur Realisierung.* expert, 2005.

WI16. WinfWiki: *Methoden und Verfahren der Aufwandschätzung im Vergleich* . Intermoves GmbH
`http://winfwiki.wi-fom.de/index.php/Methoden_und_Verfahren_der_Aufwandsch%C3%A4tzung_im_Vergleich` *(Stand: 04.02.2016)*

A

Requirements

This section contains all requirements for the developed application. It limits on the *description*, *title* and *id* of the requirements to save space.

A.1 Functional Requirements

This section describes all functional requirements of the application.

A.1.1 Database

R001 Project Database

It exists a database where all projects are stored.

R002 Project Database: Restriction

For the Bachelor Thesis the application only got a local database.

R003 Project Database: Normalization

The database is normalized in the 3 normalization form.

A.1.2 Projects

R004 Projects Overview

The application must have an overview where all projects are displayed.

R005 Filter Projects

In the projects overview the user has the possibility to filter the project properties.

R006 Sort Projects

In the projects overview the user has the possibility to sort the projects properties.

R007 Sort Projects: Standard Sort

The Standard sort method for the projects is the last edit date.

R008 Search Project

In the project Overview exists the possibility to search a project.

Project Creation

R023 Create Project

It is possible to create a new project in the application

R024 Create Project: Project Name

To create a Project a new Name must set

R025 Create Project: unique ID

All Projects have an unique ID.

R026 Create Project: Set Estimation Method

To create a Project an available estimation method must be set

R027 Create Project: Load Project Influence Factors

After an Estimation Method is set the influencing factors are loaded

R028 Create Project: Set Influencing Factors

All Influence Factors need to be set to complete the creation process

R029 Create Project: Project Phase

For the creation Process the actual Project Phase must be set

R030 Create Project: Project Properties

There has to be the possibility to add existing project documents

R031 Create Project: Development Market

The development market of the Project is selectable

R032 Create Project: Development Kind

The development kind of the Project is selectable

R033 Create Project: Process Methodology

The process methodology of the Project is selectable

R034 Create Project: Programming Language

The programming language of the Project is selectable

R035 Create Project: Platform

The platform of the Project is selectable

R036 Create Project: Industry Sector

The industry sector of the Project is selectable

R037 Create Project: Architecture

The architecture of the Project is selectable

R038 Create Project: Project Icon

The project icon of the Project is selectable

R039 Create Project: Guided Creation Process

It is possible to start a guided creation for the project which guides the user through each step

R040 Create Project: List Creation

It is possible to create a new project with a list creation which inherits all properties for the creation

R041 Create Project: Long Project Name

Project names have to be limited to 40 characters

R042 Create Project: Preselected Properties

One value must be preselected for all properties.

R043 Create Project: description

It should be possible to add a project description

R044 Create Project: Property Sorting

All Properties are sorted alphabetically descending for the selection

R045 Create Project: Suggest Estimation Method

There should be the possibility to calculate the best fitting estimation method for the project

R046 Create Project: Suggest Estimation Method - Sorting

The suggested methods are sorted after the best fitting

A.1.3 Related Projects

R009 Find Related Projects

It must be possible to find related projects.

R010 Load Components From Other Projects

It is possible to copy Components from other projects into the actual project.

A.2 Non-Functional Requirements

This section describes all non-functional requirements of the application.

R009 Design Guidelines

The application must fulfills the Design Guidelines from Google - Material Design.

R010 Save Changes Automatically

All changes in the application need to saved automatically.

B

Function Group

This section inherits an overview of all function groups of the application *MobileEstimate*.

C

Property Distances

This section introduces the distances with which each property was calculated. This is limited to the calculated distances as they are stored in the database. The calculation itself is described in section 3.4.1.

D

Entity Relation Diagram

The complete *entity relation diagram* of the *project database* is shown in this section.

E

Test Cases

This section contains all *Test Cases* with which the application was tested.

E.1 Application Tests

Start Application

Prerequisite
The application was installed on the device.
Test object
The application.
Input data
none
Procedure

1. `Click` on the application icon.

Result
The application starts in a sufficient speed.
Postcondition
The application starts and the `Project Overview` screen is visible.

Create New Project

Prerequisite
The application is opened and the `Project Overview` screen is visible.
Test object
-
Input data
none
Procedure

1. `Click` on the `'+'` icon.
2. Insert project name in the `EditText` field.
3. Insert a project description into the field.
4. `Click` on the button `'Change Project Icon'`.

5. Select an icon in the new window and click on `'save'` in the upper right corner.
6. The creation window can be seen again.
7. `Swipe` left to get to the next window.
8. The first screen with project properties is visible.
9. Select a property for the visible categories *Project Market*, *Development Kind* and *Process Methology*.
10. `Swipe` left to get to the next screen.
11. Select a property for the visible categories *Programming Language*, *Platform* and *Industry Sector*.
12. `Swipe` left to get to the next screen.
13. Select a property for the category *Software Architecture*.
14. `Swipe` left to get to the next screen.
15. This screen shows the options for *Estimation Technique* and *Influence Factor* set.
16. Select one of the available options for *Estimation Technique* and *Influence Factor*.
17. `Swipe` left to get to the next screen.
18. This screen shows an overview of all selected options for this project.
19. `Click` on `'Save Project'` in the upper right corner.

Result
A new project is created.
Postcondition
The project is visible on the `Project Overview` screen.

Create Project with List Creation Option

Prerequisite
The application was installed on the device.
Test object
The application.
Input data
none
Procedure

1. `Click` on the `'+'` icon.
2. `Click` on the expandable menu in the upper right corner.
3. Choose the option `'List Creation'`.
4. The screen shows the `'List Creation'`.
5. Name and description of the project are empty. All other options have selected a standard value.
6. Each option can be edited.
7. `Click` on `'Save Project'` in the upper right corner saves the project.

Result
A new project is created.
Postcondition
The project is visible on the `Project Overview` screen.

Cancel Project Creation

Prerequisite
The application was installed on the device.
Test object
The application.
Input data
none
Procedure

1. `Click` on the `'+'` icon.
2. Insert project name in the `EditText` field.
3. Insert a project description into the field.
4. `Click` on the button `'Change Project Icon'`.
5. Select an icon in the new window and click on `'save'` in the upper right corner.
6. The creation window can be seen again.
7. `Swipe` left to get to the next window.
8. The first screen with project properties is visible.
9. Select a property for the visible categories *Project Market*, *Development Kind* and *Process Methology*.
10. `Click` the `Back Arrow` in the upper left corner.
11. A popup message is shown which point out that this option cancels the creation process.
12. `Click` on `'Yes'`.

Result
No new project was created.
Postcondition
The `Project Overview` screen is visible.

Search Project

Prerequisite
The application inherits projects and the "Pizza Quotation Service" project is active. The Project Overview screen is visible.
Test object
none
Input data
none
Procedure

1. `Click` on the `'Search'` icon.
2. Insert *"Pizza"* in the `search` field.
3. `Click` the search icon.

Result
The search project is displayed.
Postcondition
none.

Filter Projects

Prerequisite
The application inherits projects. One *Function Point* project and one *COCOMO*. The Project Overview screen is visible.
Test object
none
Input data
none
Procedure

1. `Click` on the `'Filter'` icon in the upper right corner.
2. `Click` *Estimation Method*.
3. Select *Function Point* as method
4. Activate the filter.
5. Return to the Project Overview screen.

Result
Only Function Point projects are visible.
Postcondition
none.

Delete Project

Prerequisite
The application inherits projects and the "Pizza Quotation Service" project is active.
Test object
"Pizza Quotation Service"
Input data
none
Procedure

1. `LongClick` on the "Pizza Quotation Service" project.
2. `Click` on `'delete'` icon.
3. confirm the screen with `'yes'`.

Result
The project is deleted.
Postcondition
The project can be restored in the settings.

Transfer Estimation

Prerequisite
The relation screen is shown and inherits related projects
Test object
"Pizza Quotation Service"
Input data

none
Procedure

1. `Click` on a *related* project.
2. `Click` on `'View Estimation'`.
3. `Click` on `'Transfer Estimation'`.

Result
The estimation is transfered to the "Pizza Quotation Service" project.
Postcondition
The transfered estimation can be shown in the project.

View Projects Properties

Prerequisite
The application inherits projects and the "Pizza Quotation Service" project is active.
Test object
"Pizza Quotation Service" project
Input data
none
Procedure

1. `Longclick` on the "Pizza Quotation Service" project.
2. `Click 'Project Properties'`.

Result
The project properties are shown in a new screen.
Postcondition
The properties can be edited.

Function Point Estimation

Prerequisite
The application inherits projects and the "Pizza Quotation Service" project is active.
Test object
"Pizza Quotation Service" project
Input data
none
Procedure

1. `Click` on the "Pizza Quotation Service" project.
2. The estimation view is visible.
3. `Click` on `'Input Data'`.
4. Insert 10 in the `'medium'` field.
5. Go Back to the estimation view.

Result
The `total points`, `evaluated points` and `estimated days` were adapted.
Postcondition
none.

Start Analysis

Prerequisite
The application inherits projects and the "Pizza Quotation Service" project is active.
Test object
none
Input data
none
Procedure

1. Open the `Menu Drawer` in the Project Overview.
2. `Click` on `analysis`
3. The `analysis` screen is visible

Result
The Analysis shows all active projects.
Postcondition
none.

Open the Statistics

Prerequisite
The analysis screen is visible
Test object
none
Input data
none
Procedure

1. `Click` on `statistic` in the upper right corner

Result
The statistic screen for all project is shown.
Postcondition
none.

F

Erklärung der Kandidatin / des Kandidaten

☐ Die Arbeit habe ich selbstständig verfasst und keine anderen als die angegebenen Quellen- und Hilfsmittel verwendet.

☐ Die Arbeit wurde als Gruppenarbeit angefertigt.

Datum	Unterschrift der Kandidatin / des Kandidaten